Self-Check Laboratory Exercise 2

Decoders, Demultiplexers, Memory Address Decoding
Write VHDL code for 2-4 decoder, 3-8 decoder, demultiplexer
Verify correctness using waveforms in Model-Sim
Use LPM (Library Parameterized Modules) to create 3-8 decoder, 1-8 demultiplexer

Instructor: Professor Izidor Gertner

What to Submit:

Please post on direct private channel to Instructor just Figures with **SCREENSHOTS** of waveforms. Figure Captions should state **IN ONE SENTENCE** why the design is correct. The file name has to have your last name and title, signals have to have your last name as a prefix.
No report, nor video is required to submit.
No grade will be given for this Self-Check lab.
A Check Mark **will be assigned. The criteria used for Check Mark** (✓) A **check mark**, **checkmark** or **tick** (✓) is a **mark** used to indicate the concept "yes" (e.g. "yes; this has been verified", "yes; that **is the** correct answer", "yes; this has been completed".

1.1 1-to-2 Decoder

A decoder for the purposes of this self-check lab is a circuit that takes as input a binary string and outputs another binary string; a decoder *translates* binary information from one format to another. To become familiar with decoder's, we'll first take a look here at a simple 1-to-2 decoder. It will take one binary bit as input and output either "01" or "10". ***Please review the concept of decoder in any textbook.*** You can refer to any link on the www. To mention few:

https://www.electronics-tutorials.ws/combination/comb_5.html

https://components101.com/ics/74ls138-3-8-line-decoder-ic

1.1.1. Boolean Function

Let's name the two outputs D_0 and D_1, and the input A. This is fairly common notation for a decoder this simple, since the outputs are "data lines" and the input is the "address". Intuitively, we can describe the functionality of this circuit by saying that the address defines which of the data lines will have a "1" signal, and only one of the data lines will have that kind of signal at time. Two very simple Boolean functions adequately specify this behavior:

1. $D_1 = A$
2. $D_0 = \overline{A}$

Note that we put D_1 before D_0. This is because is the least significant bit, and we are used to seeing binary numbers arranged in this order. This will become clearer in our truth table, which is just as simple:

A	$D_1(A)$	$D_0(A)$
0	0	1
1	1	0

Figure 12. The truth table for our simple 1-to-2 decoder. Putting D_1 before D_0 makes the binary number look similar to what we are used to seeing, with the more significant bits being farther to the left.

1.1.2. Block Diagram of 1-to-2 Decoder

Let's design the Block Diagram. Intuitively, we want to split the *input* into two and invert one of the sides. This intuition is correct, and almost does the work for us in this case. The Block Diagram looks like this:

Self-Check Laboratory Exercise 2

Decoders, Demultiplexers, Memory Address Decoding
Write VHDL code for 2-4 decoder, 3-8 decoder, demultiplexer
Verify correctness using waveforms in Model-Sim
Use LPM (Library Parameterized Modules) to create 3-8 decoder, 1-8 demultiplexer

Instructor: Professor Izidor Gertner

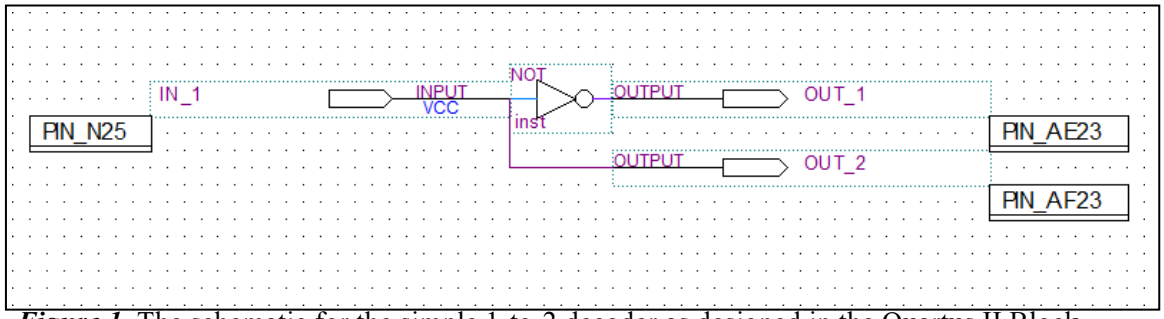


Figure 1. The schematic for the simple 1-to-2 decoder as designed in the Quartus II Block Diagram Editor. Here the inputs and outputs have different names to correspond to existing pin assignments. Note that we could easily replace the NOT gate with a NAND gate in which the inputs are bridged, but using the NOT gate makes the diagram easier to visually parse.

Self-Check Laboratory Exercise 2

Decoders, Demultiplexers, Memory Address Decoding
Write VHDL code for 2-4 decoder, 3-8 decoder, demultiplexer
Verify correctness using waveforms in Model-Sim
Use LPM (Library Parameterized Modules) to create 3-8 decoder, 1-8 demultiplexer

Instructor: Professor Izidor Gertner

1.2 2-to-4 Decoder

The previous decoder was little bit too simple to grasp the concept, so let's explore this simple 2-to-4 decoder, a circuit that does essentially the same thing, just with more bits. Now, given an input binary string of length two, we'll output a binary string of length four. Just as before, each output binary string will have only one "1" bit. This makes for four possible output values, and we know that we can map to these using two input bits.

1.2.1 Boolean Function

This time, the outputs will be D_0, D_1, D_2, and D_3; the two inputs will be A_0 and A_1. We'll use four Boolean functions to specify the design before creating a new Block Diagram:

1. $D_3 = A_0 \cdot A_1$
2. $D_2 = \overline{A_0} \cdot A_1$
3. $D_1 = A_0 \cdot \overline{A_1}$
4. $D_0 = \overline{A_0} \cdot \overline{A_1}$

Again, we reversed the outputs to match the natural ordering of binary numbers. Let's represent these functions as a truth table:

A_1	A_0	$D_3\ (A_0 \cdot A_1)$	$D_2\ (\overline{A_0} \cdot A_1)$	$D_1\ (A_0 \cdot \overline{A_1})$	$D_0\ (\overline{A_0} \cdot \overline{A_1})$
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Figure 2. Truth table for the 1-to-4 decoder. Note the diagonal of "1" values in the outputs.

So, we've shown theoretically that for each unique input binary string, we should receive a unique output binary string such that only one of the bits in each of the output strings is "1".

1.2.2 Block Diagram

With two inputs and four outputs intuition does not take us quite as far as it did in section 2.5.1., the Block Diagram for the 1-to-2 decoder. However, we've already done the difficult work of expressing our specification as Boolean functions, so we can simply "translate" those functions into a Block Diagram using the Quartus II editor. The result looks like the following:

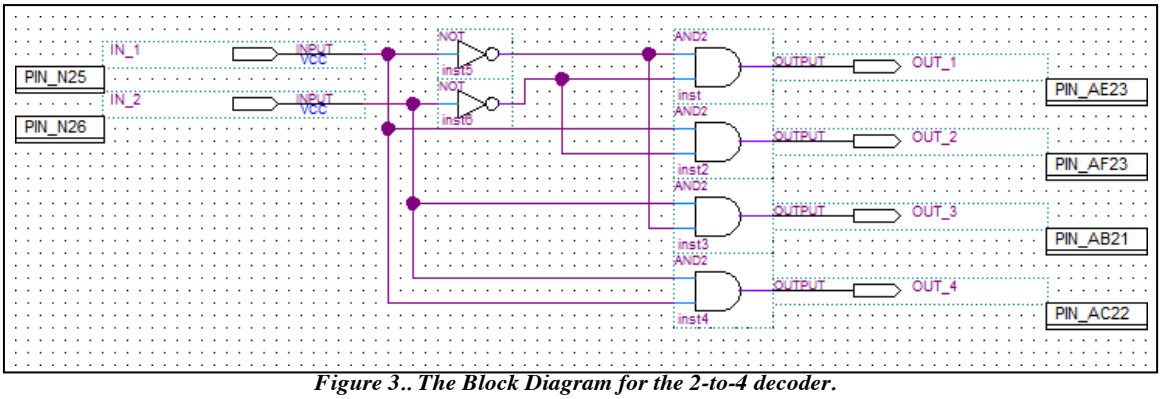
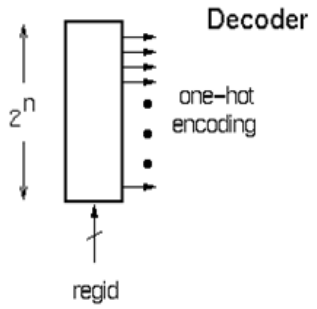


Figure 3.. The Block Diagram for the 2-to-4 decoder.

We've expressed here our Boolean functions "verbatim". We can easily replace the NOT gates with one NAND gate each and the AND gates with two NAND gates each, but this is easier to understand.
Please disregard pin symbols.

1.3 Decoder 4-to-16 Decoder



regid	sel_reg1 sel_reg0
000	00000001
001	00000010
010	00000100
011	00001000
100	00010000
101	00100000
110	01000000
111	10000000

Figure 6. Concept of decoder.

Below an example of behavioral model vhdl code for 4-to-16 decoder is shown.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity decode4to16 is
  port(
   oct : in  std_logic_vector(3 downto 0);
   dec : out std_logic_vector(15 downto 0));
end decode4to16;

architecture arch of decode4to16 is

begin
with oct select
   dec <=   "0000000000000001" when "0000",
       "0000000000000010" when "0001",
       "0000000000000100" when "0010",
            "0000000000001000" when "0011",
       "0000000000010000" when "0100",
       "0000000000100000" when "0101",
       "0000000001000000" when "0110",
       "0000000010000000" when "0111",
       "0000000100000000" when "1000",
       "0000001000000000" when "1001",
       "0000010000000000" when "1010",
       "0000100000000000" when "1011",
       "0001000000000000" when "1100",
       "0010000000000000" when "1101",
       "0100000000000000" when "1110",
       "1000000000000000" when "1111",
       "0000000000000000" when others;
end arch;
```

1.4 Memory Address Decoding using Decoder

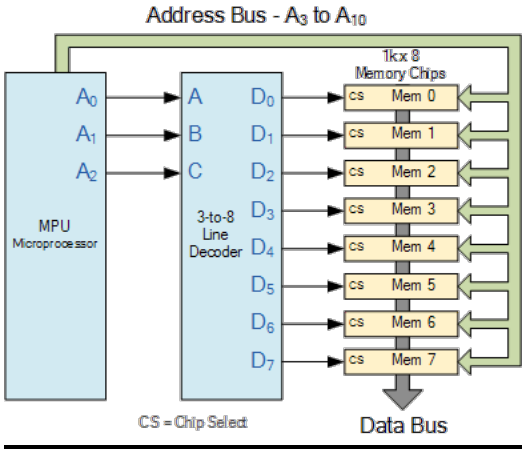


Figure 6. Memory address decoding using 3-to-8 decoder. Based on the address {A0,A1,A2} the decoder sets CSi=1 to select memory Mem i. https://www.electronics-tutorials.ws/combination/comb_5.html

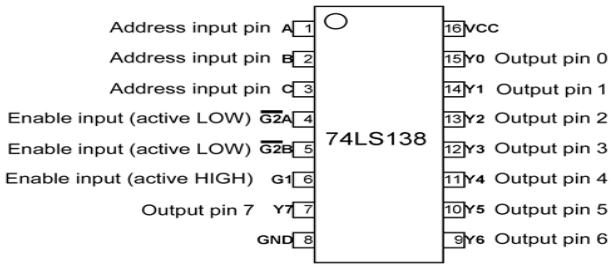


Figure 7. Example of commercial IC chip pin layout (This is old technology, just to illustrate the concept of decoder). https://components101.com/ics/74ls138-3-8-line-decoder-ic

Self-Check Laboratory Exercise 2

Decoders, Demultiplexers, Memory Address Decoding
Write VHDL code for 2-4 decoder, 3-8 decoder, demultiplexer
Verify correctness using waveforms in Model-Sim
Use LPM (Library Parameterized Modules) to create 3-8 decoder, 1-8 demultiplexer

Instructor: Professor Izidor Gertner

1.3 Demultiplexer

As the final part of this lab, let's take our first look at a demultiplexer, which is the opposite of a multiplexer. In this case, given an input signal and a selector signal, we want to direct the input to signal to one of two output ports according to the value of the selector signal. More complex demultiplexers may use more bits for the input and output ports, and will likely require more than one selector bit. However, the concept remains the same.

1.3.1 Boolean Function of 1-to-2 demultiplexer

Let's call our inputs *I* (the input signal) and *S* (the selector signal), and let's call our outputs D_0 and D_1 (the data lines). We have two outputs, so we'll use two Boolean functions just as we did in the exercise for the 1-to-2 decoder. The specification can be defined using the following functions:

1. $\boldsymbol{D_1 = S \cdot I}$
2. $\boldsymbol{D_0 = \overline{S} \cdot I}$

As usual, we'll express this is a truth table to aid us in understanding and when simulating:

I	***S***	$\boldsymbol{D_1}$	$\boldsymbol{D_0}$
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	0

Figure 4. Truth table for the 1-to-2 demultiplexer.

Once more we translate the Boolean functions, we came up with to symbols in the Block Diagram Editor. Both outputs required the use of AND gates, and the selector signal is inverted before entering the AND gate that provides the signal for the output port for the least significant bit.

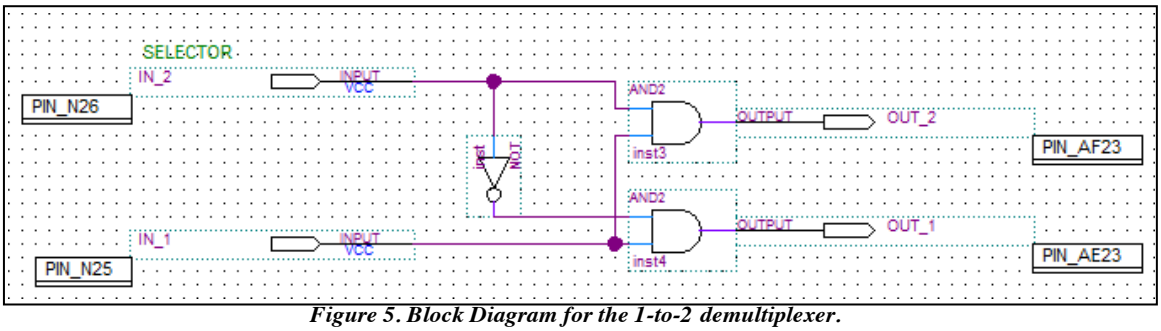


Figure 5. Block Diagram for the 1-to-2 demultiplexer.

1.3.2 Example of 1-to-n demultiplexer

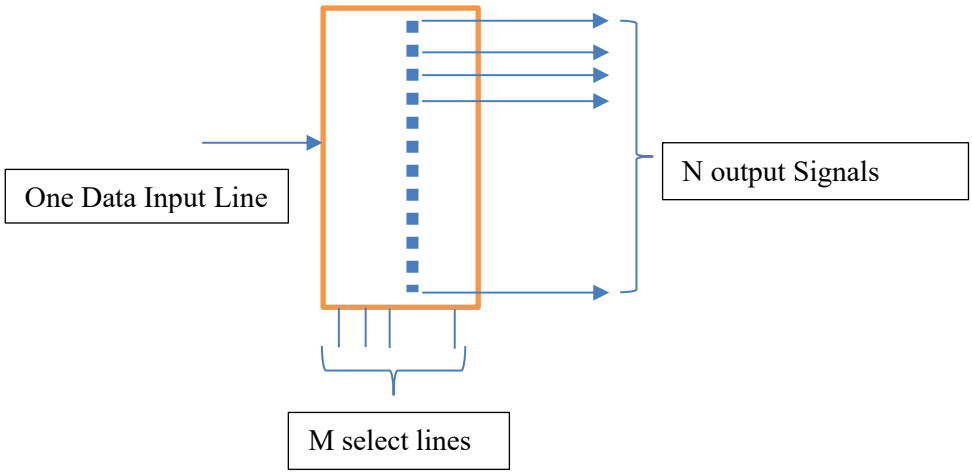


Figure 6. 1-to-N Demultiplexer: 1 input signal line, and n- output signal lines , m-the number of selection signals.

2. TASKS TO DO

TASK 2.1.

Based on Figure 3. Schematics write VHDL code for 2-to-4 decoder.

TASK 2.2 Try to find LMP module to create VHDL code for 2-to-4 decoder.

TASK 2.3. Based on decoder described in section 1.3 write behavioral vhdl code for 3-to-8 decoder.

TASK 2.4 Try to find LMP module to create VHDL code for 3-to-8 decoder.
TASK 2.5. Write behavioral VHDL model code for 1-to-8 demultiplexer.
TASK 2.6 Try to find LMP module to create VHDL code for 1-to-8 demultiplexer.

TASK 2.7 Please verify the correctness of all 4 designs using waveforms in Model_SIM. The values of output signals have to be displayed in HEX.8 bit. Each signal and input/output symbol has to have your last name as a prefix.

Self-Check Laboratory Exercise 2

Decoders, Demultiplexers, Memory Address Decoding
Write VHDL code for 2-4 decoder, 3-8 decoder, demultiplexer
Verify correctness using waveforms in Model-Sim
Use LPM (Library Parameterized Modules) to create 3-8 decoder, 1-8 demultiplexer

Instructor: Professor Izidor Gertner

3.Examples of creenshots using waveforms with Model-Sim

3.1 1-to-2 Decoder

You have to create waveforms similar to shown below in Model- Sim.

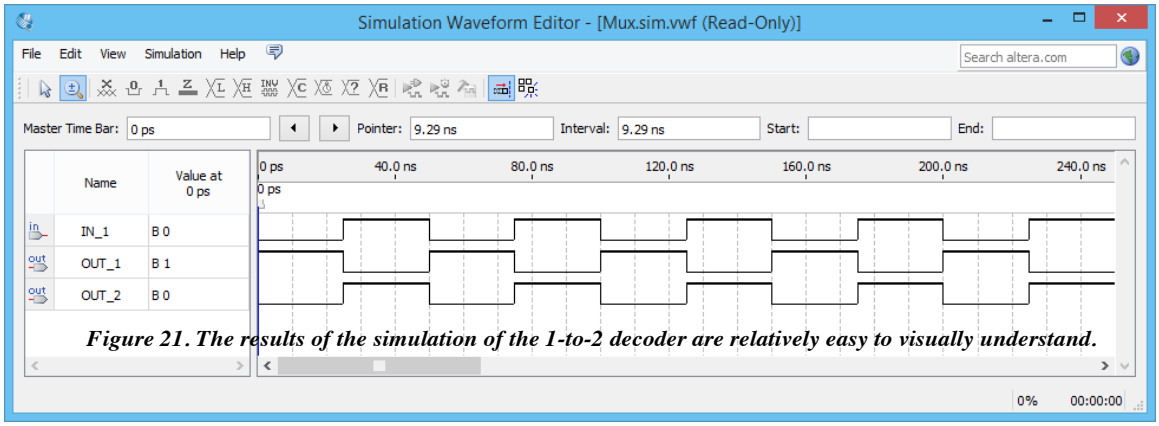


Figure 21. The results of the simulation of the 1-to-2 decoder are relatively easy to visually understand.

Figure 7. Indeed, when the address pin is set to "0", the output pins form the "01" binary string. When the address pin is set to "1", the output pins form the "10" binary string.

3.2 2-to-4 Decoder

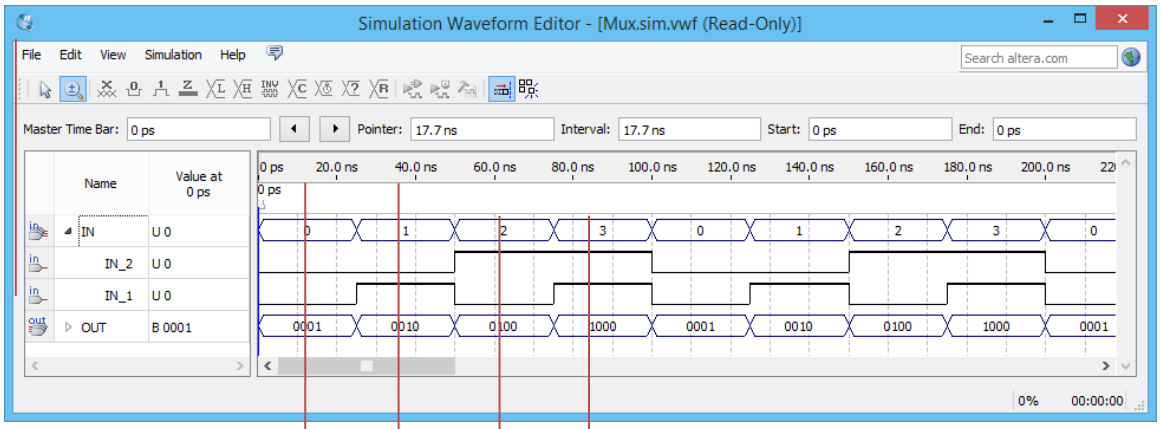


Figure 8. ***This view of the simulation of the 2-to-4 decoder makes use of group and varying radixes to make viewing easier.***

3.2 1-to-2 Demultiplexer

We again have circuit to simulate with two inputs (the input signal and the selector signal) and we therefore know that it is sufficient to oscillate the input ports with different periods, one of which is twice the other. Doing just that and running the simulation, we obtain the following result:

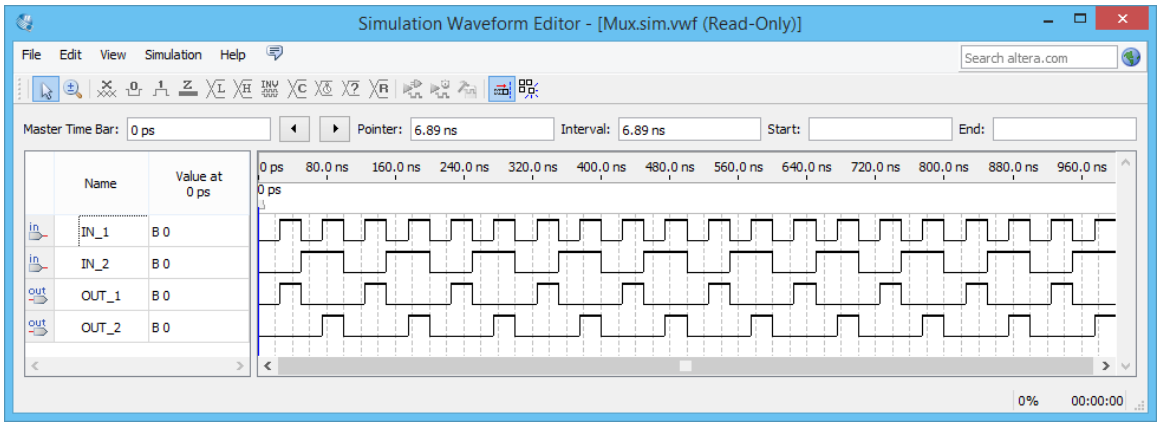


Figure 23. The post-simulation VWF file corresponding to the 1-to-2 demultiplexer.

Here, "IN_2" is the selector. This name was chosen to conform to already existing hardware pin assignments. With the input pin being oscillated at 50ns, all possible combinations are achieved by 100ns on the grid (this is why the waveforms appear to be so repetitive relative to each other). We observe that regardless of the value of the input pin, the correct output pin takes its value and both output pins never have the value of "1" .