Lab #3 Programmable Logic

Objective:

To introduce basic concepts of ROM devices and their application. To demonstrate the use of a commercial PLD design package for schematic and VHDL entry. To introduce the design process for combinational logic in a CPLD device.

Preparation:

- Read the following pages from the textbook:

  pp. 833-862 of Appendix B (Tutorial 1)
  pp. 863-877 of Appendix C (Tutorial 2)
  pp. 899-904 of Appendix E in the textbook.

If you can install the **Quartus II** *software (from the CD in the textbook) on your own machine, this can give you a significant head start as you work through the tutorial material above.*

- Familiarize yourself with the logical characteristics of the devices below.
- Read the following experiment and study the circuits as shown.
- Bring your textbook to the lab!

Devices used:

2732A	EPROM
MAX3000	EPM3032ALC44-10 CPLD

Experiment:

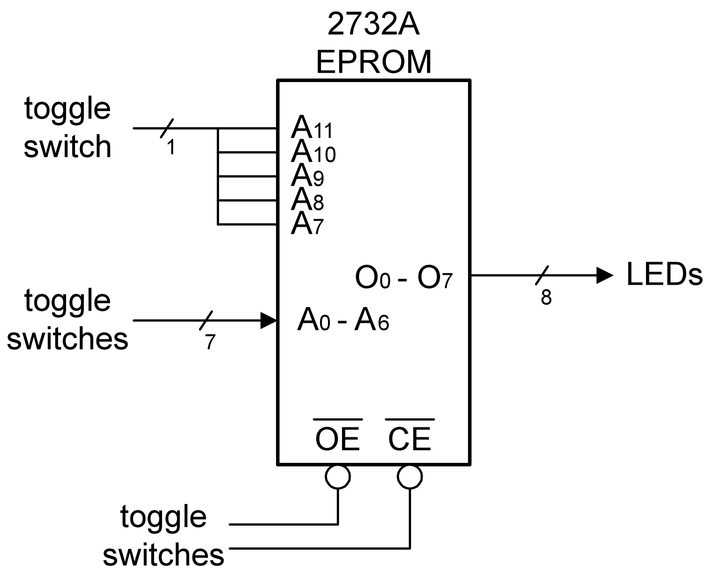
Programmable logic device (PLD) is the terminology used to represent a variety of single-chip devices that can be electronically configured to implement digital logic systems. These include read-only-memory (ROM) devices of several forms such as EPROM and EEPROM (*Flash)* memory. The first PLDs were known as programmable logic arrays (PLA), programmable array logic (PAL) and programmable logic sequencers (PLS). In recent years, generic array logic (GAL), complex PLDs (CPLD) and field-programmable gate arrays (FPGA) are more likely to be used. We will use an EPROM and a CPLD for this lab.

ROM

Read-only-memory is available in several forms including factory-programmed devices (not erasable) and those that are user-programmable and erasable (EPROM and EEPROM or FLASH memory). The most frequent use of ROM is for the non-volatile storage of microprocessor code (firmware) such as the BIOS routines in your PC or in the electronic controllers found in automotive engines and so on. However there are many other applications that use a ROM as a "look-up table" (LUT) for implementation of combinational circuits such as arithmetic functions (eg. trigonometric operations, square roots), code conversions (eg. binary to BCD), function generators, character generators for CRT displays, memory address decoding and digital image processing. The ROM we will use here is a 2732A EPROM (erasable with UV light) that is organized as 4K x 8 memory locations. It has 12 address lines (2^{12} = 4K) and 8 data lines.

1. *Look-up Table (LUT)*

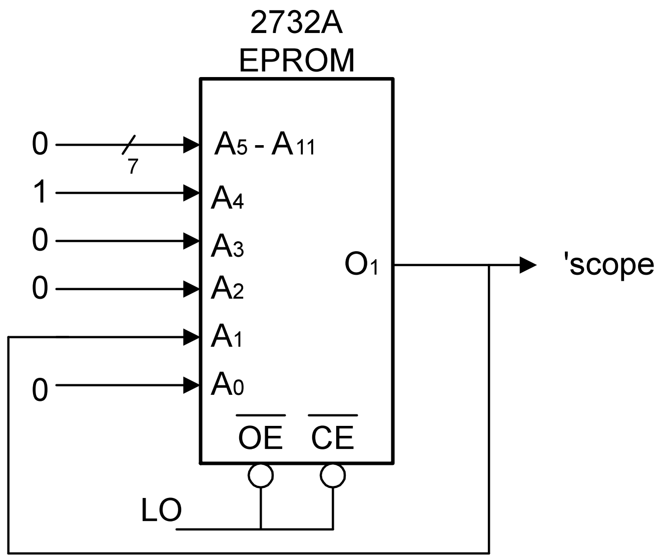
A ROM can be viewed as a look-up table for which the input is applied to the address lines and the output is taken from the data lines. By programming the ROM with a truth table, *any* combinational logic function can be implemented. Connect the 2732A as shown below. The /OE (tri-state control) and /CE lines must both be low for output data to appear on the data lines (O_0-O_7). Notice that the pre-programmed data is retained without power to the device. Connect the toggle switches exactly as shown to facilitate changes.



The 2732A devices provided in this lab contain a binary to BCD conversion LUT. A binary number is applied to the address inputs and the equivalent BCD number is read at the output. Since we have 8 output lines we can store 100 BCD numbers ranging from 00 to 99 for which we need 7 bits (A_0-A_6). The first 100 locations from address 0000 0000 0000 to 0000 0110 0011 of the EPROM are used for the look-up table that converts binary to BCD. Verify the operation of the device and the binary to BCD conversion. Note that only 100 locations of the total capacity of the EPROM have been used. That is, only 100/4096 or less than 3% !

2. *Measure Access Time*

If the /CE and /OE lines are tied LO, then one definition of access time $\mathbf{t_{ACC}}$ for memory is the delay from the application of an address input to the appearance of stable data on the outputs. Configure your circuit as shown below and observe the output from pin O_1 on the 'scope. Using this waveform, estimate the actual access time $\mathbf{t_{ACC}}$ for this particular device. Explain what you observe.



CPLDs

Programmable logic devices vary in complexity. PALs and GALs are frequently used for implementing random logic (replacing standard TTL packages), memory address decoding, bus interfacing, etc. Those that contain registered outputs can be used to implement state machines such as counters. The more complex devices (CPLD's and FPGA's) can replace large amounts of logic and can even be used to implement special purpose microprocessors.

The design of the logic to be programmed into these devices is almost invariably achieved with the use of a computer-aided design (CAD) package containing software development "tools". These tools are used for *editing* a design, *optimizing* and reducing the logic, *simulation* and *testing* and finally *fitting* the design to the target PLD. Some CAD packages can even choose the "best" PLD based on the user's design requirements. Designs may be initially entered in the form of a logic diagram using a graphic editor (this is called "schematic capture") or by using a text-based description of the hardware to be implemented. The latter is known as a "hardware description language" (HDL) and many vendors provide their own HDL development software tools (usually these are not compatible with one another!). Examples are Lattice *Synario*, Xilinx *Foundation* and Altera *Quartus II*.

Recognizing the importance of platform and architecture independence, the IEEE has developed a standard based on a widely used HDL known as VHDL. Virtually all commercially available CAD software supports the VHDL standard. That means we can design our circuits without worrying about which computer and operating system we are using or even which devices will ultimately be used for implementing our circuits!

Hardware design thus becomes an exercise in software development. The CAD software we will use here is a commercial software product called ***Quartus II*** available from *Altera* (a student version of this is on the CD with your textbook) which will compile VHDL and schematic designs for a variety of Altera CPLD's. The CPLD device we will use here is the EPM3032ALC44-10.

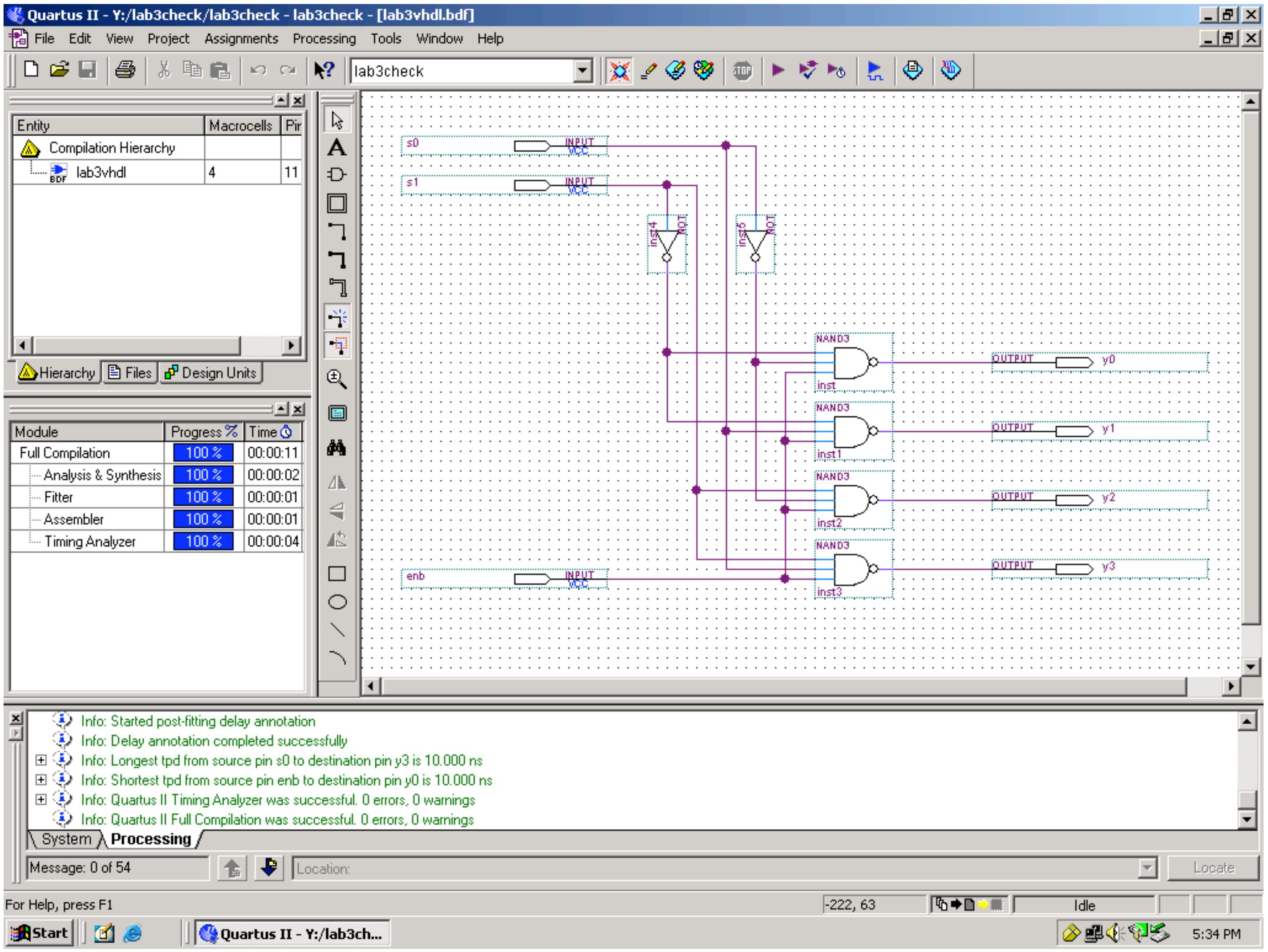
In this lab, we will design some simple combinational logic using both schematics and VHDL code.

3. Login and Setup

To begin, log in to the Windows NT workstation using the course Username and Password (the same as the web site). Double click the ***Quartus II 4.1*** shortcut on the desktop to start the CAD environment. Create a ***separate*** folder for your own work on the **Z:** drive using your student number to ensure that the folder name is unique. Create a project in this folder following the directions in Tutorial 1 (page 830) of the textbook. Of course, use your own directory and names.

4. Schematic Capture

With Tutorial 1 (section B.3.1) in the text book as a guide, enter the circuit for a 2-to-4 line decoder with LO-true outputs and a single HI-true enable (Enb) using the graphic editor. The circuit to be designed should be equivalent to:



Quartus II - Y:/lab3check/lab3check - lab3check - [lab3vhdl.bdf]
File Edit View Project Assignments Processing Tools Window Help
lab3check
s0
s1
enb
INPUT
VCC
inst4
inst5
NOT
NAND3
inst
inst1
inst2
inst3
OUTPUT
y0
y1
y2
y3
Hierarchy
Files
Design Units
System
Processing
Locate
Start
Quartus II - Y:/lab3ch...
5:34 PM

Entity	Macrocells	Pir
Compilation Hierarchy		
lab3vhdl	4	11

Module	Progress %	Time
Full Compilation	100 %	00:00:11
Analysis & Synthesis	100 %	00:00:02
Fitter	100 %	00:00:01
Assembler	100 %	00:00:01
Timing Analyzer	100 %	00:00:04

```
Info: Started post-fitting delay annotation
Info: Delay annotation completed successfully
Info: Longest tpd from source pin s0 to destination pin y3 is 10.000 ns
Info: Shortest tpd from source pin enb to destination pin y0 is 10.000 ns
Info: Quartus II Timing Analyzer was successful. 0 errors, 0 warnings
Info: Quartus II Full Compilation was successful. 0 errors, 0 warnings
```

Message: 0 of 54

Location:

For Help, press F1

-222, 63

Idle

5. Select the device

Choose **MAX3000A** as the "Family" and select **EPM3032ALC44-10** from the "Available devices" list (see Tutorial 2, section C.1.1, page 860). The device name is also written on the device.

6. Compile and Simulate

Compile your circuit as described in section B.3.2, pg 840. When your compilation completes error-free, you may simulate the logic of your design. For this example, the waveform editor (see section B.3.3) should be used to provide 8 combinations of inputs of **s0, s1** and **enb** as well as the 4 outputs **y0, y1, y2, y3**. Do not proceed until you fully understand the output of the functional simulation.

7. Program the MAX 3000 CPLD

This is described in Tutorial 3 section D.2 (pages 895-897). We are *not* using the Altera UP-1 board, but the procedure is identical for the EPM3032ALC44-10 chip.

8. Verify the chip programming

Click on the "fitter report" icon in the compiler window and scroll down to see the pin assignments that were made for your chip. You will learn later how to specify these yourself; for now just accept what the compiler chooses. ***Turn off the power*** to the prototyping board and wire your circuit according to the pinout in the fitter report. Carefully check your wiring connections before turning on the power, then test your circuit with toggle switches and LEDs to verify the truth table.

You have just generated your first custom logic device !!

Make sure you are convinced that the operation of the chip matches the functional simulation performed in part 6 of this lab.

9. VHDL design

Now, proceed in the Tutorial to section B.4 (page 846) and implement the same decoder circuit using VHDL code as shown below. Enter the VHDL code using the text editor. Compile and simulate, then implement and test in the MAX 3000 chip.

Note that the *entity* name (lab3vhdl in this example) needn't match the name of your project – but it is a good practice – and that the filename extension for VHDL text files must be *vhd*.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.ALL ;

ENTITY lab3vhdl IS
      PORT(
            s0, s1, enb           : IN    STD_LOGIC;
            y0, y1, y2, y3        : OUT   STD_LOGIC);
END lab3vhdl;

ARCHITECTURE decoder OF lab3vhdl IS

BEGIN

        y0 <= not(not s0 and not s1 and enb);
        y1 <= not(s0 and not s1 and enb);
        y2 <= not(not s0 and s1 and enb);
        y3 <= not(s0 and s1 and enb);

END decoder;
```

10. *If you have time !*

Once familiarity is gained with this entire design process, it can be achieved rather quickly. To gain additional practice, you may alter the decoder circuit to include additional enables, change the outputs to HI-true etc. Alternatively, you may repeat the entire design process for a 4:1 MUX using either the schematic entry or VHDL (or both if you have time). Program the chip and test.

Caution:

Always make sure there are no wires connected to the MAX3000 chip when programming, and always turn power off before adding wires to the chip, or disconnecting wires from the chip.