**1.0 Objectives**

Students should be able to write a script that generates binary files containing display encoding data for common-anode and common-cathode 7-segment displays. Students should also be able to program EEPROMs, understand the memory structure and addressing involved, and verify correct output based on both counting and game display modes.

**2.0 Parts List**

#

| **Quantity** | **Item** |
| --- | --- |
| 2 | White 830-point Breadboard |
| Set of | Breadboard Wire Spools | Or, you could use aPre-Cut Wire Kit |
| 1 | Wire Cutters Electronic Grade |
| 1 | Wire Strippers Electronic Grade |
| 2 | 8-Position DIP Switch SPST ([418117270908](https://www.digikey.com/en/products/detail/w%C3%BCrth-elektronik/418117270908/3174497)) |
| 2 | 2-Position DIP Switch SPST ([418117270902](https://www.digikey.com/en/products/detail/w%C3%BCrth-elektronik/418117270902/3174485?utm_adgroup=&utm_source=google&utm_medium=cpc&utm_campaign=PMax%20Shopping_Product_Low%20ROAS%20Categories&utm_term=&utm_content=&utm_id=go_cmp-20243063506_adg-_ad-__dev-c_ext-_prd-3174485_sig-Cj0KCQjwyL24BhCtARIsALo0fSB3CQfoR7PCAG1HSUgZ1s36Fn5pGQm_S2BX8wUf2TIvjAHiyHWunQIaAgqNEALw_wcB&gad_source=1&gclid=Cj0KCQjwyL24BhCtARIsALo0fSB3CQfoR7PCAG1HSUgZ1s36Fn5pGQm_S2BX8wUf2TIvjAHiyHWunQIaAgqNEALw_wcB)) |
| 2 | EEPROM Memory IC ([AT28C64B-15PU](https://www.digikey.com/en/products/detail/microchip-technology/AT28C64B-15PU/1008529)) |
| 1 | 7-Segment Display, Common-Anode ([LSHD-5601](https://www.digikey.com/en/products/detail/liteon/LSHD-5601/560008?s=N4IgTCBcDaIIwDYAMBaOBWA7OlOByAIiALoC%2BQA), [157119S12801](https://www.digikey.com/en/products/detail/w%C3%BCrth-elektronik/157119S12801/14640744?s=N4IgTCBcDaIIwFYDsc4E4DKcwA4AMcIAugL5A))  |
| 1 | 7-Segment Display, Common-Cathode ([LSHD-5503](https://www.digikey.com/en/products/detail/liteon/LSHD-5503/560009), [LDS-HTC514RI](https://www.digikey.com/en/products/detail/lumex-opto-components-inc/LDS-HTC514RI/2239244))  |
| 16 | 330 Ω THT Resistor  |
| 18 | 1 kΩ THT Resistor  |
| 1 | Breadboard Power Supply (e.g. [YwRobot MB-V2](https://static.rapidonline.com/pdf/73-4538_v1.pdf)) |

**3.0 Background**

**3.1 What is an EEPROM?**

Electrically Erasable Programmable Read-Only Memory (EEPROM) is a type of memory commonly used in digital systems to store data that must remain intact when power is removed. EEPROMs allow individual bytes of data to be written and erased through the application of voltage pulses, enabling updates without physical removal of the memory device.

Due to their byte-level erasability, ease of programming, and persistence without power, EEPROMs are often used for microcontroller-based systems, configuration storage, digital calibration, and programmable logic devices. However, they have limited write cycles, slower programming speeds, and smaller data storage capacity relative to other memory technologies like Flash or SRAM.

For this lab, you will use the AT28C64B which is an 8KB parallel EEPROM, see Figure 1 below. You will use this chip to store the display encoding data for a BCD-to-seven-segment decoder. It uses three control pins to manage read and write operations: Chip Enable (CE), Output Enable (OE), and Write Enable (WE). To read data once programmed, voltage is applied to address pins A0-A12 (you will only use A0-A8) and the data stored at that memory location is sent out to the 8 data pins, I/O0–I/O7.



Figure 1: AT28C64B pin layout. From the datasheet.

**3.2 Programming EEPROMs and Xgpro**

EEPROMs can be programmed using a variety of methods, depending on the application and hardware setup. One common method is in-system programming, where a microcontroller writes data directly to the EEPROM while the system is running. This is useful when the stored data needs to be updated dynamically, such as for user preferences or system calibration. However, in many digital logic applications—like the circuit in this lab—EEPROMs are used as static lookup tables. In these cases the contents of the EEPROM remain fixed after initial programming.

Standalone EEPROM programmers, such as the XGecu programmer and software, Xgpro, used in this lab, offer a convenient way to write data to the EEPROM before it is installed in a circuit. The XGecu programmer writes each byte of data to its corresponding address by applying the necessary signals to the EEPROM’s pins, using either an Intel HEX or binary (.bin) file as input.

Intel HEX files are text-based representations of binary data, structured in records that include address information and end with a checksum. This checksum ensures data integrity by verifying that the sum of each line’s contents, plus the checksum, equals zero modulo 256. This allows the programmer to detect and reject corrupted data.

In contrast, binary files contain raw byte-for-byte data without any structure, metadata, or error checking. They are simpler and faster to load, as they are written directly to memory starting at address 0, but they assume the data is perfectly accurate and offer no built-in protection against errors during loading. For our purposes, a simple binary file will do.

**3.3 How to install Xgpro**

If you are NOT using a lab computer, follow the steps at link below to install Xgpro on your machine. Instructions: <http://xgecu.com/MiniPro/T56_TL866II%20USER%20GUIDE.pdf>

Note that installing the drivers for this device requires administrator privileges. Additionally, the executable is not compatible with macOS. Students with macOS can use VMware to connect to VDI. Instructions for working remote: <https://it.engineering.iastate.edu/remote-computing/>

**4.0 Activity**

The circuit you will build uses 9 input bits, allowing access to 512 unique EEPROM addresses. For addresses 0 to 255, the EEPROM operates in counting mode, where 7-segment encodings for hexadecimal digits (0 to F) are repeated every 16 addresses. From address 256 to 511, the EEPROM switches to game mode, where each of the lower 8 bits directly controls one segment of the display, including the decimal point. This mode begins by storing the value 0x00 at address 256 and increments by one at each subsequent address.

**4.1 Write the Script**

Start by writing a python script to generate the binary files that will be used to program an EEPROM as a 7-segment decoder for a common-cathode type 7-segment display.

In your Python script, you will define the segment encodings and populate an array using two for loops. This array should then be written to a binary file, and it’s good practice to include a print statement confirming that each file was successfully generated. Before moving onto the next step, have the TA check your code for the proper encodings.

**4.2 Setup the Software**

To begin, open the XGecu software from the Windows Explorer. Ensure that the programmer is connected to the computer via USB. Verify that the correct programmer model is selected, as shown in Figure 2. Refer to the label on the back of the programmer to confirm the model number, it will likely be T48. If a device selection popup appears, confirm or select the correct model. Depending on previous usage, this popup may not appear.

Next, click “Select IC” in the upper-left corner of the software window. In the device selection menu, search for and select the AT28C64B EEPROM, as shown in Figure 3. Keep all other settings as default.



Figure 2: Confirm the correct programmer model is selected.



Figure 3: Select IC.

**4.3 Load the Binary File**

Once the XGecu software is properly configured, load the binary file. From the menu, select “Load”. Next, in the popup menu, select “BINARY” as the file format and navigate to the correct file using the “Browse” option. Finally, hit “OK” and the main display should be updated with the correct hexadecimal encodings as seen in Figure 5. Before proceeding, have the TA check that the loaded table is correct.

 

Figure 4: Load the first binary file.



Figure 5: Common-cathode display encoding.

**4.4 Program the EEPROM**

After ensuring that the address encodings were properly loaded, program the EEPROM. From the main menu, select “Program”. A popup window will appear that will show the “Location in Socket”, as seen in Figure 6. It is important that the software has recognized the correct EEPROM and programmer as the location in the socket varies.

Flip the lever on the programmer so it is in the upright position, perpendicular to the body of the programmer. Insert the EEPROM as shown, pay attention to the orientation of the chip and its first pin. Once the chip is placed, flip the lever back down and hit the “Program” button in the popup menu. The software will check that the location of the chip is correct before attempting to program. Once it has successfully programmed the chip, you will see a success message like in Figure 6.



Figure 6: EEPROM location in socket and programming success message.

**5.0 Build the circuit**

**5.1 Place Components on the Breadboard**

Place a 2-position DIP switch on the far left side of your breadboard. This will contain the select bit on its right switch. To the right of it, place an 8-position DIP switch. These switches represent the address inputs to the EEPROM (bits A0 to A7). Place the AT28C64B EEPROM in the center of the breadboard. On the far right side, place your 7-segment display, leaving room for the power supply which will be placed later. An example layout is shown in Figure 7.

**5.2 Wire the Switches**

Connect the top side of each switch to Vcc. Connect the bottom side of each DIP switch through a 1kΩ pull-down resistor to ground. This ensures a stable LOW signal when switches are open. Note that this is pictured in Figure 7 with the resistors on top of wires, however, it will be easier to keep the resistors below any wires placed later. This change is reflected in the image in Figure 8.

**5.3 Power and Configure the EEPROM**

Connect the pin 28 of the EEPROM to Vcc and pin 14 to ground. Ground the no connect (NC) pins and the A9–A12 address pins (Pins 21, 23, 24, and 2) since only 8 address lines are needed for this circuit. Also, ground both $\overline{CE}$ and $\overline{OE}$ (pins 20 and 22) to enable reading from the chip. Lastly, connect $\overline{WE}$ (pin 27) to Vcc to disable writing and prevent accidental programming during circuit use.

**5.4 Connect DIP Switches to EEPROM**

Assign switch 8 from the larger DIP to A0, this will be the least-significant bit. Continue connecting the remaining DIP switches in order down to switch 1 on the large DIP, which is the most-significant bit in the address. The mapping is:

* Switch 8 → Input bit 0 → A0 (Pin 10)
* Switch 7 → Input bit 1 → A1 (Pin 9)

…

* Switch 2 → Input bit 8 → A6 (Pin 7)
* Switch 1 → Input bit 7 → A7 (Pin 6)

Switch 2 on the smaller DIP will act as our select bit. When low, numbers will be displayed in counting mode. When this input is high, the display will be in game mode where each segment of the display corresponds to a switch. This select bit should be connected to pin 25, which is the address input A8.

**5.5 Connect EEPROM Outputs to 7-Segment Display**

Start by grounding the common-cathode pins of the 7-segment display, pins 3 and 8. Next, connect EEPROM data outputs I/O0–I/O7 to the appropriate segments on your 7-segment display based on your answer to question 1 in the pre-lab.



Figure 7: Completed example implementation.

**6.0 Testing**

Begin testing your circuit in counting mode by setting the select switch to 0. Use the DIP switches to toggle through different input combinations. You should observe the 7-segment display cycling through hexadecimal digits (0–F). Only the four least significant bits should affect the output—the four most significant bits should have no effect. Test as many combinations as needed to confirm the behavior is consistent.

Once counting mode is verified, switch to game mode by setting the select switch to 1. In this mode, each of the eight least significant bits directly controls a segment on the display. Start by toggling the least significant bit; it should control the top segment of the display. As you move upward through the bits, the segments should light in a clockwise pattern, ending with the center segment and then the decimal point.

After confirming this behavior, demonstrate to the TA that the binary inputs 0111, 1000, 1011, 1111, and 0000 produce the expected segment outputs on the display in counting mode. Then, switch to game mode and toggle each bit to ensure that every segment responds correctly.

**7.0 Common-Anode vs Common-Cathode Displays**

Common-anode and common-cathode 7-segment displays require opposite signal logic to illuminate segments. In a common-cathode display, all cathodes are connected to ground, so segments are turned on by applying a high signal to the corresponding anode. In contrast, a common-anode display connects all anodes to a positive voltage, and segments are turned on by applying a low signal to the corresponding cathode. As a result, the EEPROM encodings for common-anode and common-cathode displays are different.

This lab does not require you to encode a common-anode decoder, but before proceeding, explain to the TA how you might modify your python script to produce the proper binary file for a common-anode display.