Contents

[Purpose 1](#_Toc94551233)

[Abstract 1](#_Toc94551234)

[Support Components 2](#_Toc94551235)

[M27C256B EPROM 2](#_Toc94551236)

[Initial Read: 3](#_Toc94551237)

[Intro 3](#_Toc94551238)

[Harness 3](#_Toc94551239)

[Initial Connections: 5](#_Toc94551240)

[Automated address and data management via Raspberry Pi GPIO 5](#_Toc94551241)

[Python Code 6](#_Toc94551242)

[Testing M27C256 Harness 7](#_Toc94551243)

[Dumping the EPROM 8](#_Toc94551244)

[Erasing the EPROM 8](#_Toc94551245)

Purpose: To explore various memory technologies including:

* UV erasable EProm
* Electrically erasable (14 volt Vpp, 5 Volt Vcc)
* Electrically erasable 5 volt

Abstract:

This activity was the first attempt at understanding and working with various memory technologies. The initial experiments focused on the M27C256B EPROM chip. This chip was found in a package of electronic parts collected over a span of 40 years by a relative. The second set of chips included the W27E040 EEProm purchased through Amazon. These chips are an older technology and used. They were delivered with data still contained on the chips: More on that later. Finally, this set of activities will include programming and operation of a current technology 5 volt EEProm, the AT28C256 EEProm capable of reads and writes using only 5 volt supplies.

The labs will progress from the oldest technology through the newest. In addition to reading and writing the different chips, additional support components will be introduced. This includes, Python, Raspberry Pi GPIO, Breadboards, Serial in parallel out (SIPO) IC’s ad Parallel in Serial out (PISO) IC’s. Additionally, debugging techniques for both hardware and software will be detailed throughout the various labs.

Support Components:

For data reads and writes the lab will proceed from manually setting control and addressing using a 5V source and reading the data at specific locations using LEDS to using raspberry pi 2 and 3’s to supply power, control and addressing signals while leveraging the raspberry pi GPIO pins to read the data from the memory chips.

As the labs progress from the UV erasable M27C25B chip leveraging 15 bit addressing through the W27E040, leveraging 19 bit addressing, additional IC components will be used to manage IO to and from the chips. This will include the 4021 CMOS 8-stage Shift register as a PISO component supporting data reads. The labs will also leverage several 4094 CMOS 8 stage Shift and store Bus Register used as a SIPO component.

Other tools used for development and testing include a voltmeter an SDS 1202X-E digital Oscilloscope. An assortment of resistors and LED devices will also be used for pull-up, pull-down, data indicators, and voltage dividers resister arrays. Power sources include the 5 volt pins of the Raspberry Pi GPIO, 5 Volt sources from modified USB charging cables (photos included), and a 28 Volt variable voltage source to support erasure and writing of the W27E040 chips. Finally, a dual wavelength, UV-A 315 nm, and, UV-C 280 nm, UV lamp will be used to erase the M27C256B EPROM chip.

M27C256B EPROM:

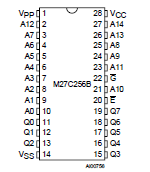


Figure 1

The M27C256B is a UV erasable 15 bit EProm with an 8 bit data output. From the ST Electronics data sheet dated april 2006:

“The M27W256 is a low voltage 256 Kbit EPROM offered in the two ranges UV (ultra violet erase) and OTP (one time programmable). It is ideally suited for microprocessor systems and is organized as 32,768 by 8 bits.

The M27W256 operates in the read mode with a supply voltage as low as 3V. The decrease in operating power allows either a reduction of the size of the battery or an increase in the time between battery recharges.

The FDIP28W (window ceramic frit-seal package) has a transparent lid which allows the user to expose the chip to ultraviolet light to erase the bit pattern. A new pattern can then be written to the device by following the programming procedure. For applications where the content is programmed only one time and erasure is not required, the M27W256 is offered in PDIP28, PLCC32 and TSOP28 (8 x 13.4 mm) packages.”

## Initial Read:

Intro: The first step taken with this chip was verification of operation. This required collection of manufacturer data to include the chip datasheet. The data sheet gives the pin out and voltage/signal levels necessary for read, write, and erase operations.

Setup of the initial read took three phases:

1. Initial chip verification
2. Initial read (manual)
3. Full dump

For each of the above stages the wiring and harness verification steps will be detailed.

Harness:

Throughout these activities standard breadboards will be used and are shown below:

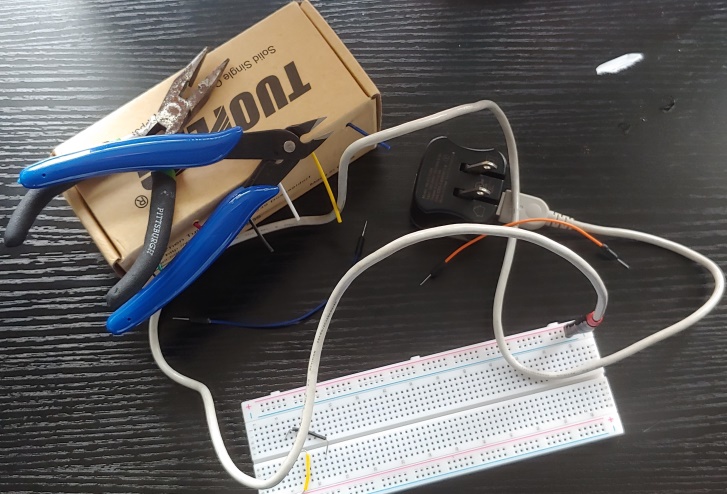


Figure 2

The tools used shown above to operate the chips are:

1. Needle nose pliers
2. Wire cutters
3. 22 gauge solid core wire (multicolor) for custom connectors (black and yellow seen on breadboard and box dispenser)
4. Assortment of factory produced breadboard male-female and male-male connectors (orange connector)
5. Modified USB cord and USB power supply (5V power supply)
6. Assorted resistors and Light Emitting Diodes (LED’s) to see data and manage voltage and current

Most of the components are self-explanatory but the USP cable will be described an a little more detail. The USB interface for most USB cables (excluding fully implemented USB-C) is a 4 wire connector. Two of the wires are power and ground (red, black) and the other two wires are signal. For this activity, the USB connector end was removed exposing the four wires. The two signal wires were cut back and taped down as they are not used. The power and ground wires were soldered to standard breadboard pins with plastic separators:

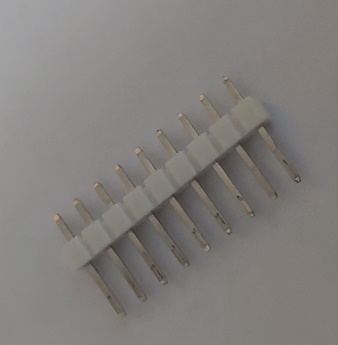


Figure 3

The plastic separators align the pins to be spaced the same distance as the breadboard connectors. In this case, two pins were broken from a line a nine, keeping the two connected. These two pins were then soldered to the power and ground wires from the USB cable and wrapped with scring wrap to protect the connections as illustrated below:



Figure 4

### Initial Connections:

Figure 4 below illustrates the initial connections for the M27C256B chip. Using the pin out information from figure 1 the power, control, address and data pins are connected. In figure 4 the poser and ground rails are connected to pins 14 (ground) and 28 (Vcc = 5V). All address pins are connected to ground through 500 ohm resisters as pull down resisters ensuring well defined “low” values for all address pins. All data pins are connected to the positive side of the “data” LED’s. The low side of each LED is connected to ground through 500 ohm resisters. The resistors are used to both configure the default state of pins (pull-down, or pull-up) and also as current limiters.

Figure 4 also has a free yellow connector connected to Vcc on one side. This connector can be used to look at different memory location by connecting it to various address pins bringing them high. When the M27C256B chip was purchased, it contained data indicating the chip was purchased used. By bringing each of the 15 address pins high different addresses can be accessed.

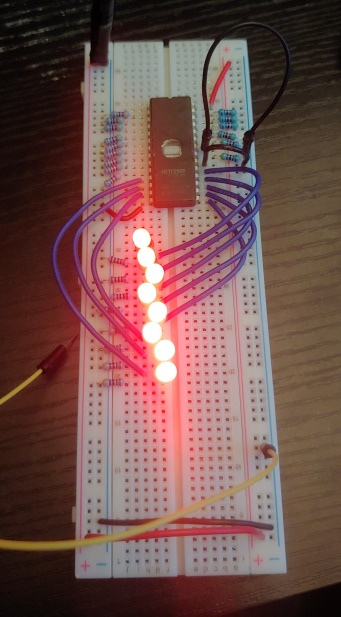


Figure 5

In figure 4, the UV reprogramming window can be identified. This window allows light to cover the die. If that light has a UV component it will set all bits in the chip to logical 1. Programming only converts logical 1’s to logical zero (open). This will be discussed later in the document.

Automated address and data management via Raspberry Pi GPIO:

Access all memory addresses and data requires 15 connections for addresses and 8 for the 8 data pins . The Raspberry pi has sufficient IO ports to support this. For this automated dump of the entire M27C256B, 33 pins will be used on the Raspberry Pi GPIO interface pictured below.

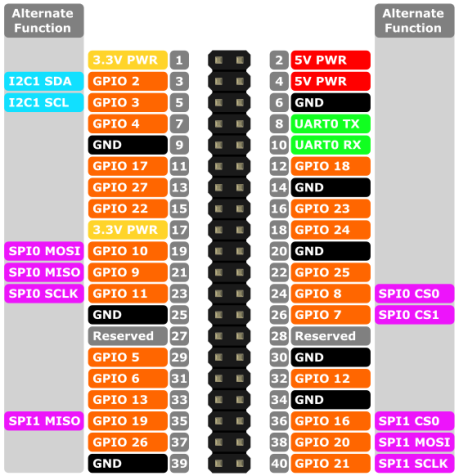


Figure 6

For this stage of reading the chip, the Raspberry Pi will also be used to provide power and ground. For this phase, the control pins, chip enable and output enable will be tied to ground since these are enable low and do not need to be changed for this read operation.

Python Code:

**import RPi.GPIO as GPIO # import RPi.GPIO module**

**from time import sleep # lets us have a delay**

**GPIO.setmode(GPIO.BOARD) # choose BCM or BOARD**

**Ai = [11,13,15,16,18,22,29,31,32,33,35,36,37,38,40] # set address pin GPIO\_11 = A0, ... GPIO\_40 = A14**

**Bi = [3,5,7,8,10,19,21,23]** **# set address pins GPIO\_3 = B0, … GPIO\_23 = B7**

**for y in range(len(Bi)):** **# initialize Raspberry Pi pins for addressing**

**GPIO.setup(Bi[y],GPIO.IN)**

**dump = 0x00**

**outbin = open("chip.dmp", "wb")**

**count = 0**

**for y in range(len(Ai)):** **# initialize Raspberry Pi pins for data input**

**GPIO.setup(Ai[y], GPIO.OUT, initial=0)**

**for y in range(15):** **# run through address pins for hardware verification**

**GPIO.output(Ai[y], 1) # set each address pin high in order A0 on chip to A14**

**print("bit " + str(y)) # Identify the bit position just activated**

**var = input("hit key when ready for next bit") # wait for keyboard input to activate the next pin**

**try:**

**for x in range(0x0,0x7ffff):** **# walk through all addresses sequentially 0 -(2^15 - 1)**

**for y in range(len(Ai)): # for each address, set the appropriate address pin values**

**GPIO.output(Ai[y],((x>>y)&1))**

**for bits in range(8): # read associated data bits and pack then in a output byte**

**dump = dump | (GPIO.input(Bi[bits]) << bits)**

**outbin.write(dump.to\_bytes(1,byteorder='little'))** **# write the data out to the output file**

**dump=0**

**except KeyboardInterrupt: # trap a CTRL+C keyboard interrupt**

**GPIO.cleanup()**

**GPIO.cleanup() # cleanup GPIO pins**

The code above has four major components. The first is the mapping of GPIO pins to address and data bits. There are 15 address bits mapped to GPIO bits as defined in the array Ai. The data bits are mapped to GPOI pins as defined in the array Bi.

The second section configures the GPIO pins as output for the address pins. When set to logical 1, the address pins will pull up the address pins on the M27C256 chip a logical 0 on a GPIO pin will leave the associated pin on the M27C256 chip at logic zero, low because of the pull down resister in the breadboard tied to each address pin. The data pins are managed similarly except the pins are configured as input. One the address pins are set, the data pins will hold the values of the 8 bit word at that address location. So, the address pins are output pins for the GPIO interface and the data pins are input pins for the GPIO interface

The third section is used to sequentially walk through hall the address pins and ensure that the each GPIO pin and connected to the correct address pin. This will be descried in more detail below

The fourth section is the main code for walking through hall address sequentially to dump the EPROM. An index ranges from 0 – (2^15 -1). The index is decoded into 15 address bits and set to output through the GPIO address pins. The next call reads the associated data pins and packs the 8 bits into a byte for output. Finally the output byte is written to the output file

The last section is cleanup. The python write buffer is flushed pushing all remaining data to the output file and the output file is closed. The last call sets the GPIO bits to their default state.

Testing M27C256 Harness:

As shown in figure 7 below:

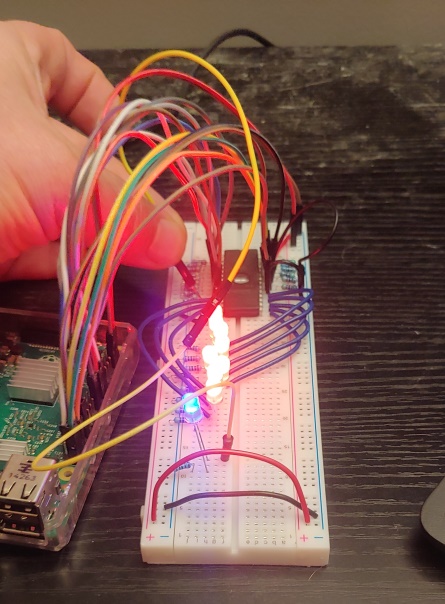


Figure 7

Prior to dumping the EPROM both the python code and the wiring of the harness needs to be verified. For the address and data pins the connections between the address array positions that map to specific GPIO\_pins must be connected to the correct address pins. To verify this, the array is stepped through and enabling each address pin in the address sequence from A0 to A14. At each step the most recently enabled address pin is connected to a test diode, the blue diode in the above picture to ensure the correct pin active and no other pins are illuminated.

Using the same process the data packing can be verified. In this case rather than looping through all memory locations, several locations with different data values are selected. The data values can be read off the red LED’s which indicate the data bits. The python code then takes the same string of zero and ones and packs them into the output word. The output word is then verified against the LED values.

Dumping the EPROM:

Erasing the EPROM:

The M27C256 is a Ultraviolet light erasable EPROM. The specifications require a UV source with a frequency in the area of 250 nanometers. UV erasable will erase with a broad range of environments including direct sunlight. However the farther from the specified UV frequencies or sunlight requires longer exposure. For this activity, a UV lamp with frequency in the specified range is available and is



Figure 8

pictured in figure 8. It is a BLAK-RAY mineralight with windows for both long-wave (315 nm) and short wave (280 nm). The screen behind UV lamp is a section of the chip dump showing a very rich data set. The shortwave uv light will be used for the erasure as depicted in figure 9

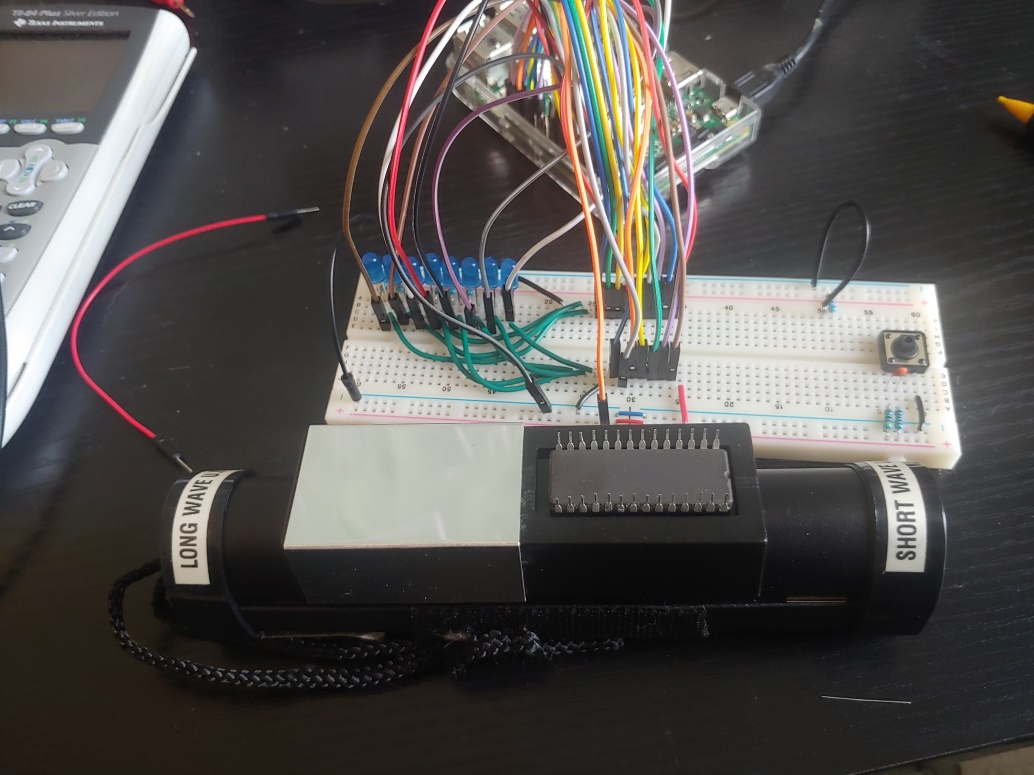


Figure 9

The M27C256B chip was placed over the UV light, 280 nm for one hour. The chip was then placed back in the test harness and all addresses were accessed and the data bits were all 1, the data LED’s never went out demonstrating that the Chip has been successfully erased.