Contents

[Purpose 2](#_Toc95054696)

[Abstract 2](#_Toc95054697)

[Support Components 2](#_Toc95054698)

[M27C256B EPROM 3](#_Toc95054699)

[Initial Read: 3](#_Toc95054700)

[Intro 3](#_Toc95054701)

[Harness 4](#_Toc95054702)

[Initial Connections: 5](#_Toc95054703)

[Automated address and data management via Raspberry Pi GPIO 6](#_Toc95054704)

[Python Code 6](#_Toc95054705)

[Testing M27C256 Harness 8](#_Toc95054706)

[Dumping the EPROM 8](#_Toc95054707)

[Erasing the EPROM 9](#_Toc95054708)

[W27E040-12 11](#_Toc95054709)

[Pinout 12](#_Toc95054710)

[CD4021-BE Static Shift Register: 13](#_Toc95054711)

[CD4021-BE operation: 13](#_Toc95054712)

[EEPROM dump code: 15](#_Toc95054713)

[W27E040-12 Data: 16](#_Toc95054714)

[Data identification: 16](#_Toc95054715)

[FOS data: 17](#_Toc95054716)

[Decode 17](#_Toc95054717)

[Erasing the W27E040-12: 18](#_Toc95054718)

[Erase Connections: 18](#_Toc95054719)

[Voltage Dividers Up Close 19](#_Toc95054720)

[Programming the Chips: 21](#_Toc95054721)

[Programming the M27C256B: 21](#_Toc95054722)

[Chip Specifications for Write Operations 21](#_Toc95054723)

[Data and Code for Write Operation 22](#_Toc95054724)

[Verification of the write operation: 23](#_Toc95054725)

[Programming the W27E040-12: 25](#_Toc95054726)

Purpose: To explore older memory technologies requiring special tools or DC voltage levels for programming and erasure. The chips under investigation including:

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

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 were used parts. They were delivered with data still contained on the chips: More on that later.

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 Parallel In Serial Out (PISO) component supporting data reads. The labs will also leverage several 4094 CMOS 8 stage Shift and store Bus Register used as a Serial In Parallel Out (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 divider 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:

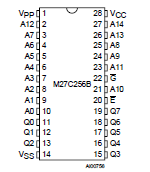


Figure 1

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:

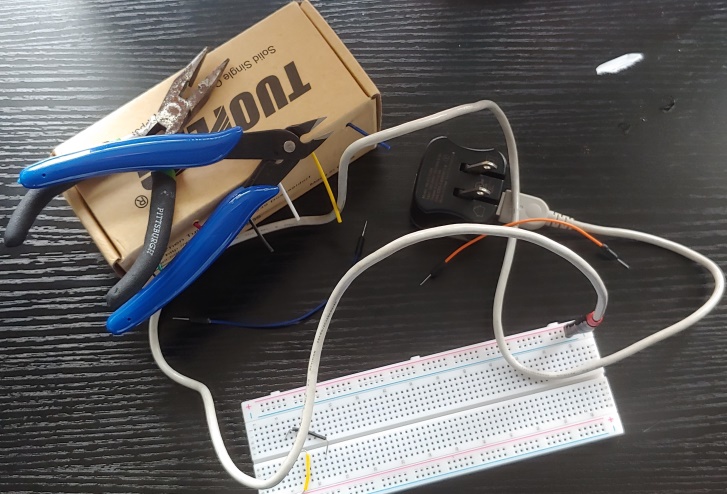


Figure 2

The tools used and 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:

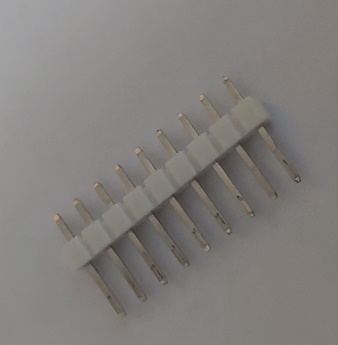


Figure 3

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 shrink wrap to protect the connections as illustrated below:



Figure 4

### Initial Connections:

Figure 5 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 positive and ground rails are connected to pins 14 (ground) and 28 (Vcc = 5V) of the M27C256B. 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 aquired, it contained data indicating the chip was used. By bringing each of the 15 address pins high different addresses can be accessed.

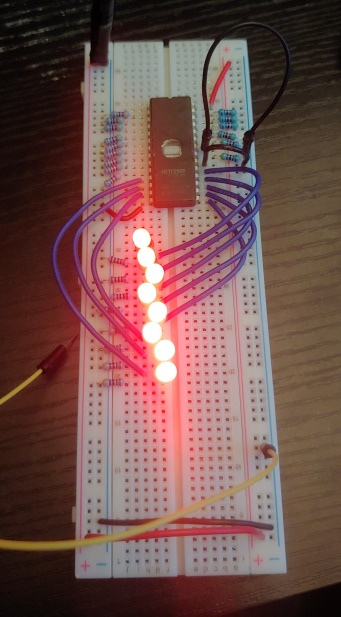


Figure 5

In figures 1 and 5, 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:

Accessing 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.

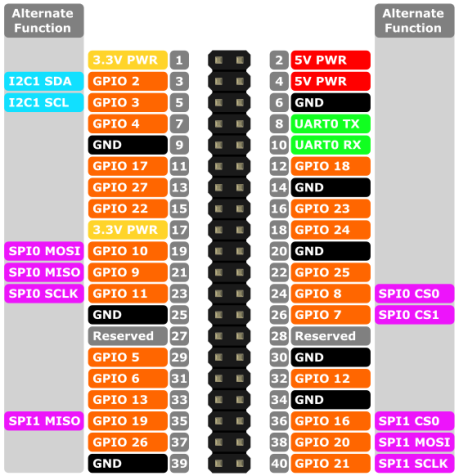


Figure 6

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 GPIO 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 be pulled up to a logical 1. If the GPIO pin s set ot a logical 0, it 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. Once 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 a wirin test section and is used to sequentially walk through all 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 all 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:

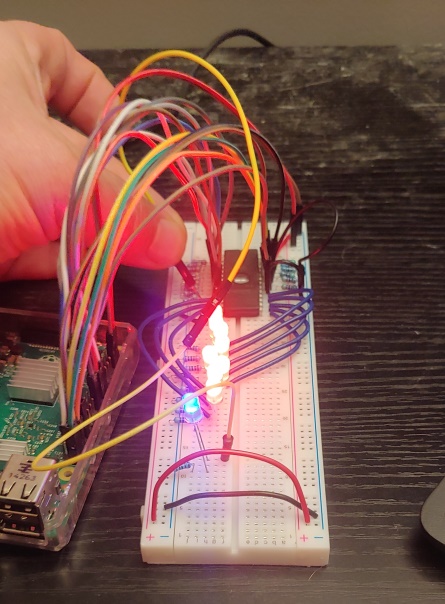


Figure 7

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:

For dumping the EPROM the code section to test out the harness can be commented out and the main loop is set to run through (0 -> 2^15-1) memory locations. For the read, the LED’s were left connected to monitor the data flow. Below is a dump of the first few pages of the M27C256B chip.



At present, no analysis of the dump has been completed

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

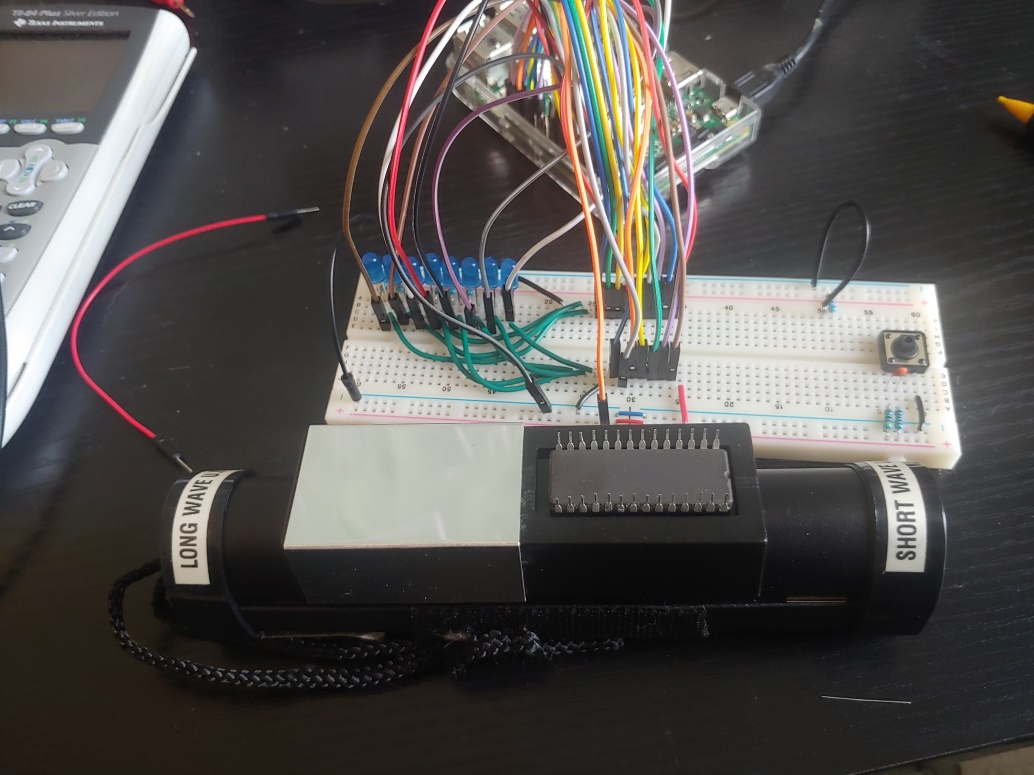


Figure 9

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.

W27E040-12:

The work done to connect, dump, and clear the M27C256B will be extended to the next memory chip. The W27E040-12 chip is an electrically erasable chip. It has two main differences from the M27C256B. The first is the address space. The W27E040-12 is a 19 bit address device requiring 4 additional address lines. Including the 8 data lines, there are not enough I/O ports on the RPi GPIO interface. To expand on the number of ports available to the GPIO, a Parallel - In - Serial – Out (PISO) shift register will be added to the collection harness.

The second difference with the W27E040-12 is the method of erasure. It is a EEPROM meaning it is Electrically erasable. To erase the chip requires a variety of voltage levels including a 14V input to the programming pin. To perform this, a 24V variable DC power supply will be used with a set of resisters to build 3 voltage divider circuits supporting voltage levels needed to erase the chip.

To support connectivity between the chips, breadboard, and RPi, and GPIO extender board will be used. The complete harness is shown below in figure 10. The view in figure 10 illustrates the connection between the RPi and the GPIO extension board which plugs into the standard breadboard connections. Using standard 22 gauge breadboard wiring for the majority of the connectios makes for a much cleaner harness. In general, yellow will be used for addressing, blue for data, red for power, black for ground, and white for PISO control and serial data pin connections.

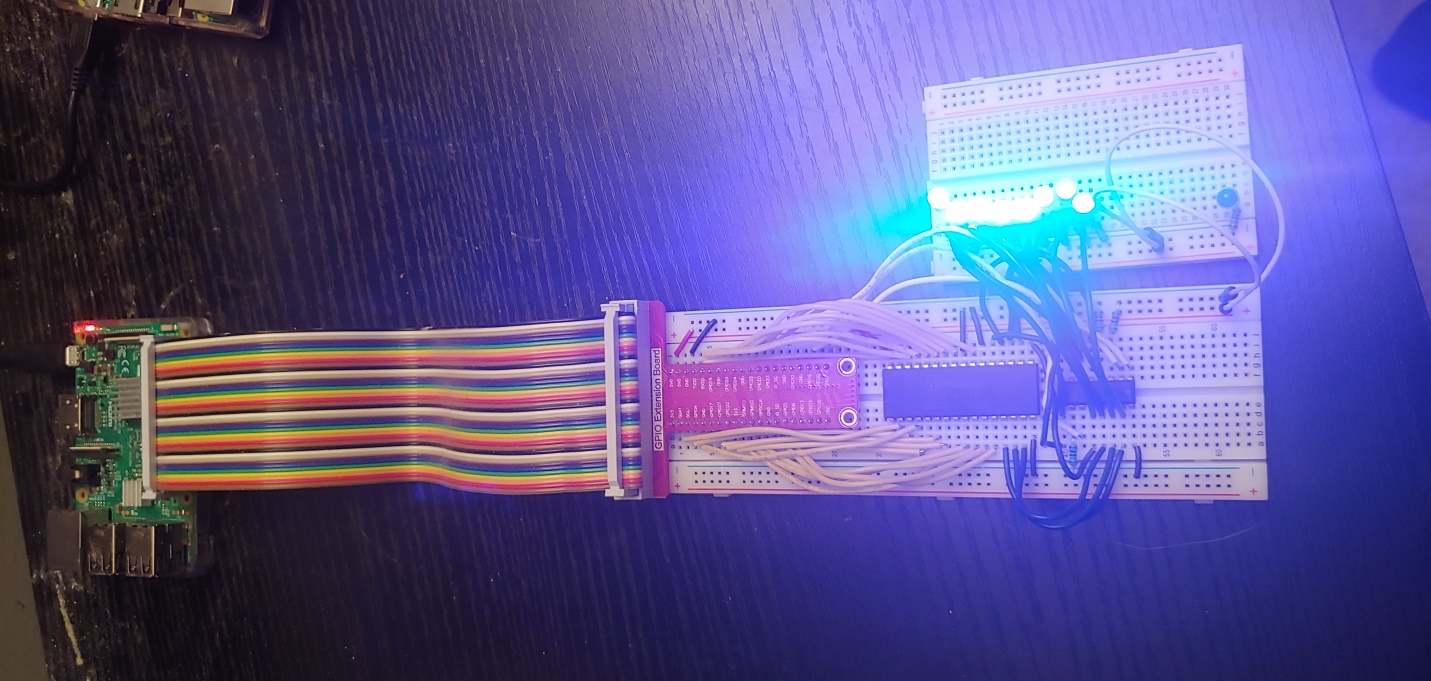


Figure 10

Figure 11 shows a close up of the bread board including the addressing and data connections. Additinoally, the 4021 PISO chip

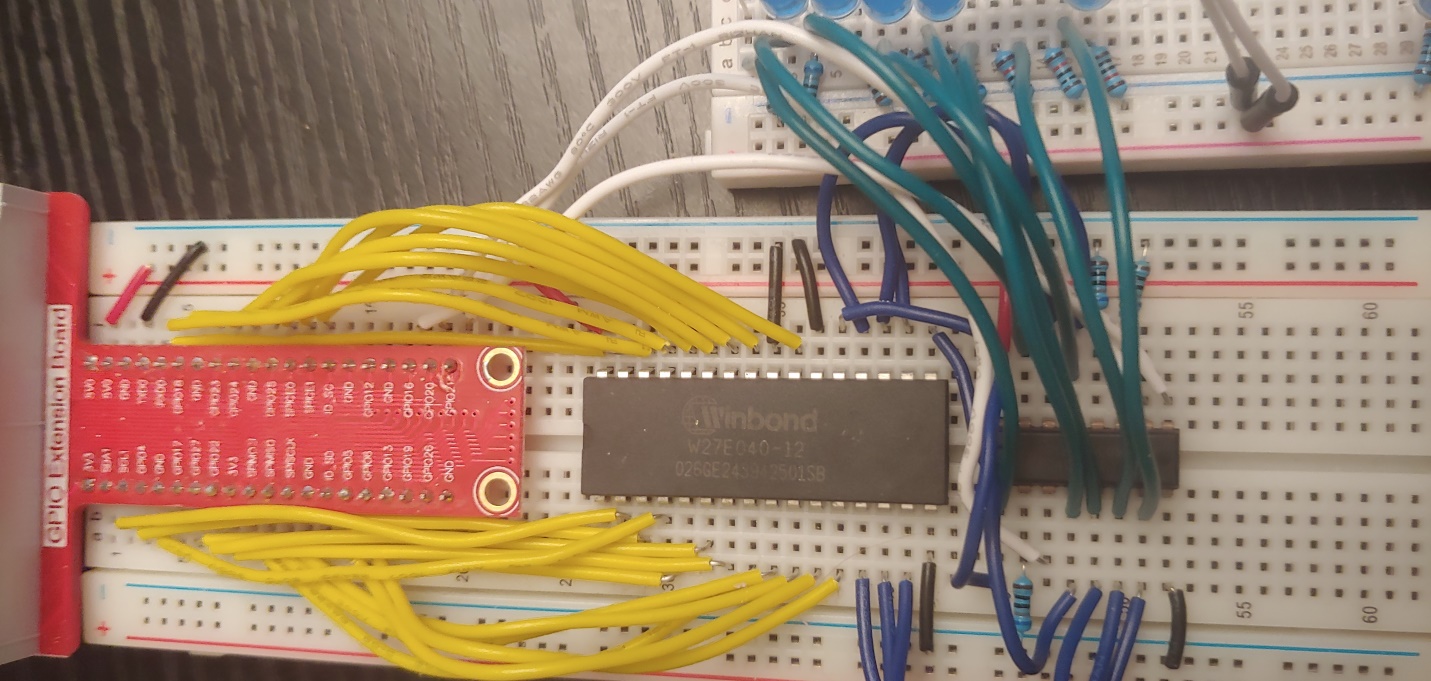


Figure 11

Figure 11 gives a closeup view of the harness showing the addressing and data connections, hardwired control pin connections such as output enable and chip enable, and the green wires pulling off the data bits for the LED bank to observe data flow out of the EEPROM and into the PISO

Pinout:

W27E040: Figure 12 illustrates the pin out of the W27E040 EEPROM. It shares many of the characteristics of the EPROM above. The main differences are in the address space, 19 address pins instead of 15, and a new pin Vpp . This pin is used for erasure of the chip and programming. The procedures for both will be described below.

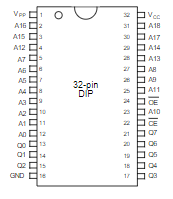


Figure 12

CD4021-BE Static Shift Register: Figure 13 illustrates the pinout of the CD4021 PISO shift register. This shift register will be used to pull data from the EEPROM. The EEPROM provides data as 8 bit words requiring 8 pins to collect. Given the additional address pins needed to access the EEPROM data, the RPi does not have enough pins to collect the data and manage control pins. To solve this problem the PISO chip will be used to collect the 8 data bits in parallel and the RPi will clock out the data from the PISO serially. The control and data pins needed to clock in the data only require 3 RPi pins. In the pinout below, the pins listed as PI-1 through PI-8 are the data pins and are the parallel inputs to the PISO. These will be connected to the data pins of the W27E040 and do not consume any RPi pins. The RPi pins will control the clock, parallel/serial control, and Q8. The Q8, Q7, and Q6 pins are serial out registers. This effort will only use the high order Q8 pin. This accounts for the three pins needed by the RPi to clock in data

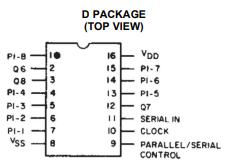


Figure 13

## CD4021-BE operation:

Operation of the CD4021-BE is fairly straight forward.

W27E040

CD4021-BE

Q0 -> PI-1

Q1 -> PI-2

Q2 -> PI-3

Q3 -> PI-4

Q4 -> PI-5

Q5 -> PI-6

Q6 -> PI-7

Q7 -> PI-8

Q8 -> RPi Serial in

Parallel/Serial control -> RPi Parallel/Serial control

Clock -> RPi clock

Figure 14

Figure 14 is a simplified connection diagram illustrates the data flow from the EEPROM to the PISO shift register and finally to the RPi. The 8 data lines of the EEPROM are connected to the 8 parallel in pins of the PISO. The values of the data pins are always available to the PISO. Internally, the PISO can be thought of as 8 storage units. When the Parallel serial control is brought high, and the clock is strobed, the values presented to the 8 inputs of the PISO will be latched into the 8 storage units. From the Q8 pin the value of the 8th storage register can be read when the serial /parallel control is held low. On every clock pulse with the parallel/serial control pin low the contents of the storage unit at position “n” will be replaced with the value of the storage register and position “n-1”. So after the data is latched into the PISO, Q8 holds the high order bit. After a clock cycle, Q8 holds the data that was in storage unit 7, another clock cycle; Q8 has what was in storage unit 6, and so on. In this way, the data can be clocked out of the PISO and read by the RPi.

The code snippet used to clock data out of the PISO is listed below:

def get\_data():

outbyte = 0

GPIO.output(Latch,1)

GPIO.output(Clock,1)

GPIO.output(Clock,0)

GPIO.output(Latch,0)

for i in range(8):

bit = GPIO.input(Data)

outbyte = outbyte << 1

outbyte |= bit

GPIO.output(Clock,1)

GPIO.output(Clock,0)

return outbyte

Figure 15

This Python code snippet has three variables Latch, Clock, and “outbyte”. For each data event there are 4 GPIO events. The first sets the latch to one. Latch corresponds to the Parallel/Serial control. It enables the parallel read on all 8 data lines. The next command, GPIO.output (Clock, 1), strobes the clock which completes the load into the PISO storage registers (D-Flip-flop). The next two commands take clock to zero for the negative transition. The GPIO.output (Latch, 0) puts the PISO devices in serial mode to begin the read.

Next is the loop, remember, the Q8 pin of the PISO has the value of the high order bit of the 8 bit data word. Each iteration will shift “outbyte” 1 bit and then add in the read data bit to outbyte. Finally, the clock pin is cycled causing all the data bits in the PISO to shift 1 toward the high bit (as wired). After 8 steps through we have the day byte in the correct order.

## EEPROM dump 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

import pdb

Ai = [31,29,23,21,19,15,13,11,18,22,26,24,7,16,12,5,3,10,8]

Latch = 36

Clock = 38

Data = 40

GPIO.setup(Latch,GPIO.OUT,initial=0)

GPIO.setup(Data,GPIO.IN)

GPIO.setup(Clock,GPIO.OUT,initial=0)

outfile = open("w27e040.dmp", "wb")

def get\_data():

outbyte = 0

GPIO.output(Latch,1)

GPIO.output(Clock,1)

GPIO.output(Clock,0)

GPIO.output(Latch,0)

for i in range(8):

bit = GPIO.input(Data)

outbyte = outbyte << 1

outbyte |= bit

GPIO.output(Clock,1)

GPIO.output(Clock,0)

return outbyte

index = 0

for y in Ai:

print(index,Ai[index])

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

index = index +1

# pdb.set\_trace()

try:

for x in range(0x0,0x7ffff):

for y in range(19):

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

outbyte = get\_data()

outfile.write(outbyte.to\_bytes(1,byteorder='big'))

outfile.close()

except KeyboardInterrupt: # trap a CTRL+C keyboard interrupt

GPIO.cleanup()

GPIO.cleanup()

Figure 16

This is a little cleaner code then what was used in the previous dump. A function has been defined to interact with the PISO chip to pull data bytes. The main address loop only needs to break out each address bit and use them to set the W27E040 address lines. Once set, the get\_data function is called, and the data byte is written to the output file.

## W27E040-12 Data:

### Data identification:

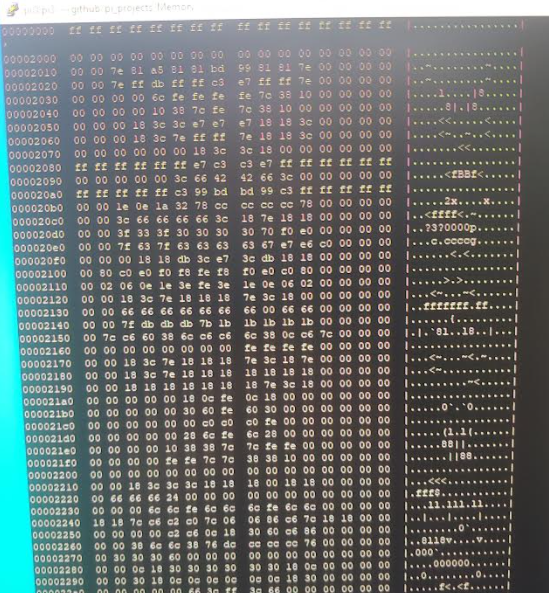


Figure 17

Figure 17 illistrates a screen shot of a hex dump of the data that was on one of the W27E040 chips. The data was evaluated on the RPi using the linux command:

Hexdump -C W27E040.dmp

Where the filename W27E040.dmp was hardcoded in the code illustrated in figure 16. After doing a google search on the data line :

7e 81 a5 81 81 bd 99 81 81 7e

there was a hit on a website that described the formatting of FON files used in the computer game called “FALLOUT” at <https://falloutmods.fandom.com/wiki/FON_File_Format>. A screen shot of the file format with the coding description is given in figure 18.

### FOS data:

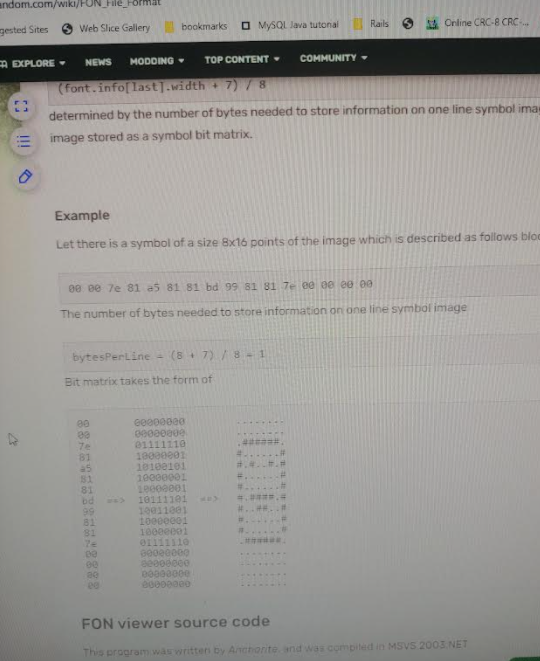


Figure 18

Following the encode/decode procedure as stated the python code listed in figure 19

Decode:

import sys

from time import sleep

args = sys.argv[1:]

data = open(args[0],"rb")

data.seek(0x2010)

def build\_string\_line(inbyte):

outstring = ""

littlec = ""

for i in range(8):

bit = (inbyte>>i)&1

if (bit == 1):

littlec = "\*"

else:

littlec = " "

outstring += littlec

return outstring

for i in range(1024):

bytes = data.read(16)

for j in range(16):

img\_line = build\_string\_line(bytes[j])

print (" " + img\_line)

sleep(0.5)

data.close()

Figure 19

Data images: Figure 20 illustrated decoded images from lines in the dump file. Each image is built up of 16 contingious bytes. From the file where each byte is represented in binary and stacked one on the other. A zero gets a space character and a 1 gets a “\*”.



Figure 20

There was no real point in investigating the data other then to see what was on the used EEPROM chips. Also, the data doens not necessarily have anything to do with the game Fallout. It is just the source used to identify the data and encoding scheme.

## Erasing the W27E040-12:

### Erase Connections:

The W27B256 differs from the M27C256B in that it can be erased electrically. There is no need to use devices like UV lamps. The W27B256 is a little chalanging because it does not use 5V and control pins to erase or program the chip. Specifically, a 14 V source is required to erase the W27B256. Figure 21 illustrates the harness used to erase the W27C256 chip. To erase the chip the following pin assignments must be made:

* Vpp = 14 V
* Vcc = Vce = 5V
* CE \_bar = VIL (0.8 volts or lower but above ground)
* OE\_bar = VIH (2.0 V or above but lower then Vcc)
* A9 = VHH(14V)
* A0 = VIL
* All other address pins to be held a VIL
* All data pins held at VIH

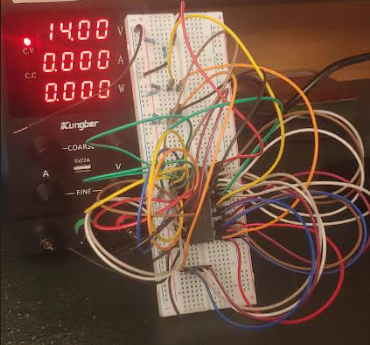
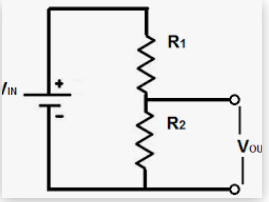


Figure 21

Voltage Dividers Up Close:

The complication for erasing data was the need to supply several different voltage sources. This was done using the concept of voltage dividers. A simple voltage divider is illustrated in figure 22.

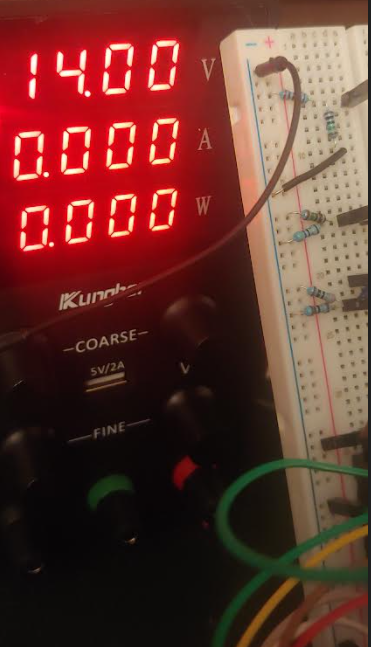


This circuit was replicated 3 times to get the needed voltage levels for erasing the chip. The resisters used were are listed in Table 1:

Vout can be determined by the simple relation Vout = Vin(R1/(R1+R2)). Figure 22 illustrated a close up of the voltage dividers. They are the three resister sets in the upper right.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Divider target voltage | Input voltage | R1 | R2 | Actual voltage |
| 14 | 14 | none | none | 14V |
| Vcc = VCE = 5V | 14 | 150 ohm | 270 ohm | 4.88 V |
| VIH | 14 | 47 ohm | 150 ohm | 3.34 V |
| VIL | 14 | 10 ohm | 330 ohm | 0.4 V |

Table 1



Top resistor is 10 ohm, bottom is 330 ohm. Measured Voltage between resistors was 0.4V

Top resistor is 150 ohm, bottom is 47ohm. Measured voltage between is 3.3V

Top resistor is 150 Ohm, botton is 270 ohm. Voltage measured between the resistors was 4.88 V

Figure 22

Once configured. The chip is connected and the Erasure takes a few miliseconds. Each chip was erased and then put in the dump harness to verify that the data was all 1’s. In figure 22, the red rail is 14 volts, the first set of resistors is for 5V (4.8 actual)

# Programming the Chips:

## Programming the M27C256B:

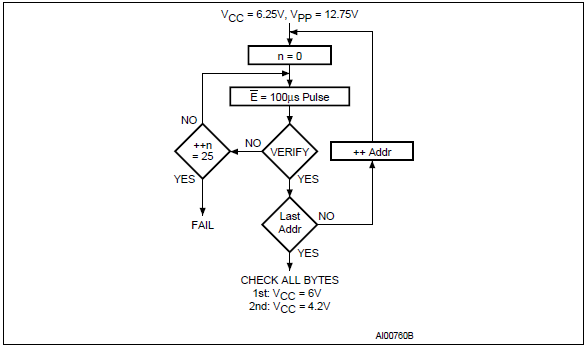


Figure 23

Figure 23 illustrates the operation flow taken from the chip manufacturer data sheet to program the M27C256B chip. In words, this flow will walk through a write operation in two main phases. The first is the the initial write. For all writes the output enable E\_bar pin is kept high since we are performing a write operation and E\_bar is active low. Once the address and data values are presented to the address and data pins and the time to stabilize is passed, the chip enable which is held high is pulsed low. This enables the chip and the write operation. Where the flow goes next is a function of the Verify which will read the byte just written and verify that it is correct. If it is, the address is steped to the next address with the associated data. If the verification fails, the flow will pass to the left which will continue to write by pulsing the chip enable and will continue write attempts until it succeds or 25 attempts have been made.

Chip Specifications for Write Operations:

To do the write, the M27C256B has specific voltage requirements that differ from the read operations voltage Vcc, VIL VIH. To program the EPROM, the programming pin Vpp needs to be set to 12.75 volts. The specifications also call for Vcc to be 6.25 volts. Since the Rpi can only supply 5 volts through the GPIO a second power source will need to be used. Care must be taken on the second power supply. If you supply greater then 5V to and pin on the GPIO, it will be damaged. Figure 24 illustrates the writing harness. There are three components, the Rpi, Chip+breadboard, and the Kungber DC power supply. The DC power supply is configured with V- tied to ground. To ensure a consistant voltage referance across the three components, the ground from the DC power supply and the GPIO ground are tied together on the Chip breadboard.

Prior to power everything on, it is important to ensure that Vpp is isolated from all GPIO pins. To do this an ohm test was run between the Vpp pin and every other pin and GPIO pin.

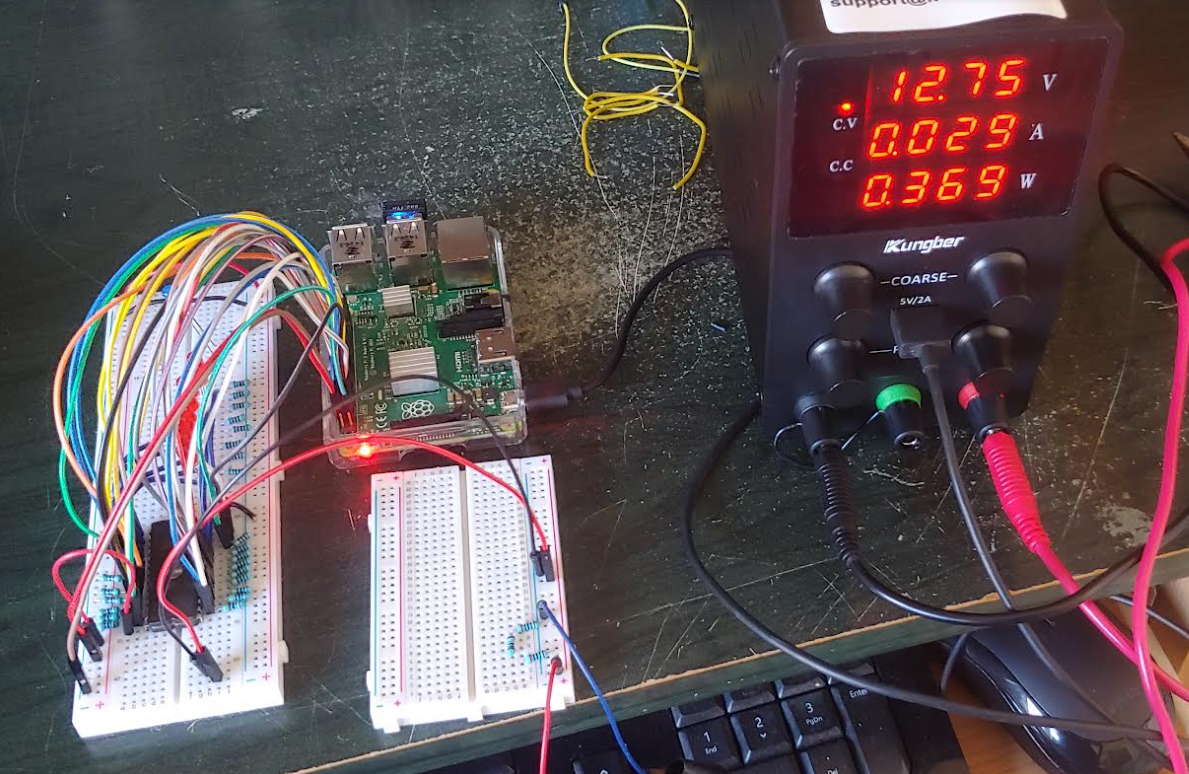


Figure 24

Even though the specification calls for 6.25 volts on Vcc for write operations, this write attempt was done with Vcc at 5V pulled from the GPIO. This was done to minimize the chance of over driving the GPIO pins. For this test, the only positive voltage pulled from the DC power source is the 12.75 programming voltage connected to Vpp. It turns out that this is sufficient for programming the chip.

Data and Code for Write Operation:

To minimize the number of GPIO pins needed to test out the write operations, the 8 data pins are tied to the low order 8 address pins. This means the memory location data assignment will equal the memory location address. If all works the zeroith address will contain 0x00, address 1 will contain 0x01, .., address 254 will contain 0xfe, and address 255 will contain 0xff.

Python code for the write operation:

Figure 25 illustrates the code for writing data to the M27C256B. Common with previous sets of code the GPIO pins are set. The code does not need GPIO pins to set the data bits to the M27C256. The bit assignments are hardwires as follows A0 -> D0, A1 ->D1, A2->D2, .. , A7->D7. There are two additional control pins assigned, The chip enable and output enable are set in the code. The output enable pin is set to high disabling output and setting up the data pins for input. From the specifications, the write operation takes place with Vpp = 12.75 volts, the address and data pins are set and ready, the output enable bit is set to high, and the chip enable bit is strobed. The chip specification calls for the chip enable pin to be taken low for at least 100 ns. This is seen in the last loop section in the “try” part of the code.

A second simplificatin was made for this test. Rather then do a read verify for each memory location and perfom the additional writes till the data is properly set, this code will write each memory location 10 times. The contents of the chip will be verified after the write operation is complete.

Finally, the write operation will only write to addresses 0 through 255.



Figure 25

### Verification of the write operation:

To verify the write operation, the data pins need to be connected and the hardwire connections between the address and data pins used for the the above write operations need to be removed. This is essentially the same python code used for the original dump but here, only address zero through 0x3ff will be read since address above 0xff were not programmed. The commented code was added for development to enable single stepping and debugging but was commented out in the final run

**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]**

**Bi = [3,5,7,8,10,19,21,23]**

**for y in range(len(Bi)):**

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

**dump = 0x00**

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

**count = 0**

**OutputEnable = 24**

**ChipEnable = 26**

**GPIO.setup(OutputEnable, GPIO.OUT, initial=0)**

**GPIO.setup(ChipEnable, GPIO.OUT, initial=0)**

**for y in range(len(Ai)):**

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

**try:**

**for x in range(0x0,0x3ff):**

**for y in range(len(Ai)):**

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

**for bits in range(8):**

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

**outbin.write(dump.to\_bytes(1,byteorder='little'))**

**# print("address " + str(x) + "Byte Read " +hex(dump))**

**# var = input("hit key")**

**dump=0**

**sleep(0.05)**

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

**GPIO.cleanup()**

**GPIO.cleanup()**

Figure 26

The hex dump of the verify operation is listed in figure 2

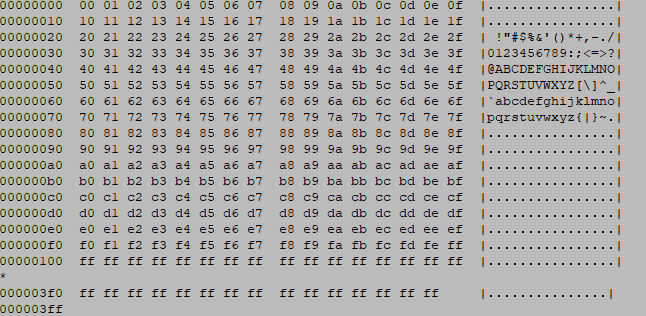


Figure 27

## Programming the W27E040-12:

The W27E040-12 is very similar to the M27C256 for programming. From the datasheet for the W27E040-12:

“Program Mode

Programming is performed exactly as it is in conventional UVEPROMs, and programming is the only

way to change cell data from "1" to "0." The program mode is entered when VPP is raised to VPP

(12V), VCC = VCP (5V), CE = VIL, OE = VIH, the address pins equal the desired address, and the input

pins equal the desired inputs “

Rather then repeat the work done above for the M27C256. The activities will move to a EEPROM that supports reads and writes using only 5V power supply.