Search for

**nint64's Project Blog**

**an Electronic Engineering Student**

☰ **Menu**

# EEPROM Programmer using an Arduino – Part 1 (The SST39SF040 and Planning)

mikemint64 Electronics Jul 29, 2018Aug 15, 2018 5 Minutes

I have seen many different ROM dumpers and programmers for retro game systems such as the Nintendo GameBoy, this got me wondering how I could achieve the same thing with an Arduino as most of the programmers use the same chip as the Arduino boards to interface with the cartridge. I will be attempting to use an Arduino to program a parallel EEPROM / NOR Flash which can then be used to replace the ROM in older games such as those for the Gameboy, NES or the SNES.


*SST39SF040 and AM29F040B*

The EEPROM / NOR Flash I am working with is the **SST39SF040**, however, the process would be the same for the SST39SF010A and SST39SF020A as they are the same flash chips of different memory sizes.

I imagine this would be compatible with most parallel flash EEPROM chips as they seem to have similar pin-outs and command sequences such as the **AM29F040B**, a common EEPROM used to replace the GameBoy cartridge mask ROM.

# The SST39SF040 NOR Flash Chip

Firstly we should begin by having a look at the pins available for our use and what each of them does. We can find all this information inside of the data-sheet (http://ww1.microchip.com/downloads/en/DeviceDoc/20005022C.pdf). This data-sheet also contains information on the other variations of this chip so make sure you are looking at the correct information.

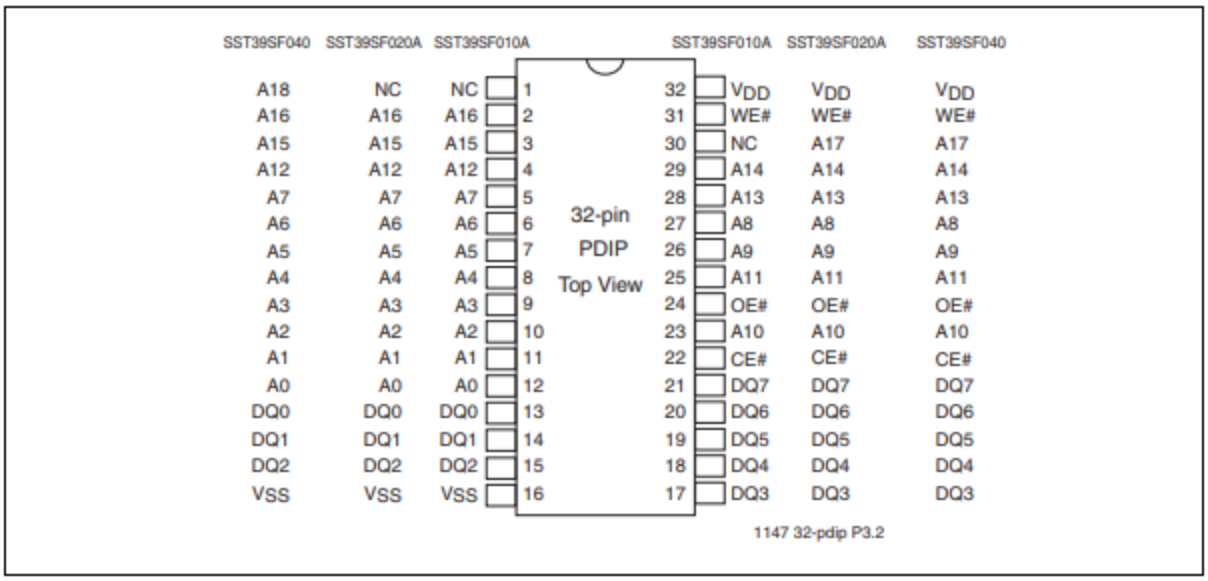



**Figure 4:** Pin Assignments for 32-pin PDIP



Here is a quick description of the function of each pin, the data-sheet also provides its own table on each pins function.

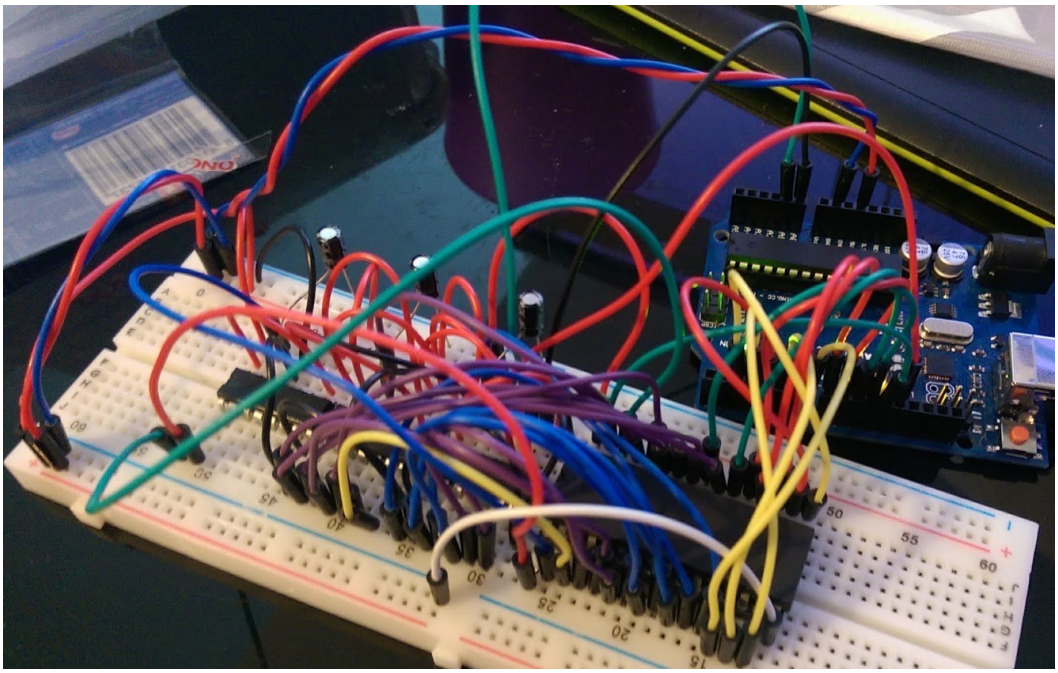
- **A0 – A18 : the address pins that will be used to select the address of where to read/write a byte of data**
- **DQ0 – DQ7 : the data pins that will either output the 8-bits(1-byte) of data from the MSB at DQ7 and the LSB at DQ0, they also are used to input the byte of data during writing data or commands**
- **VDD : 5V DC supply**

- **VSS : Ground**
- **OE# : Output-Enable input signal**
- **WE# : Write-Enable input signal**
- **CE# : Chip-Enable input signal**

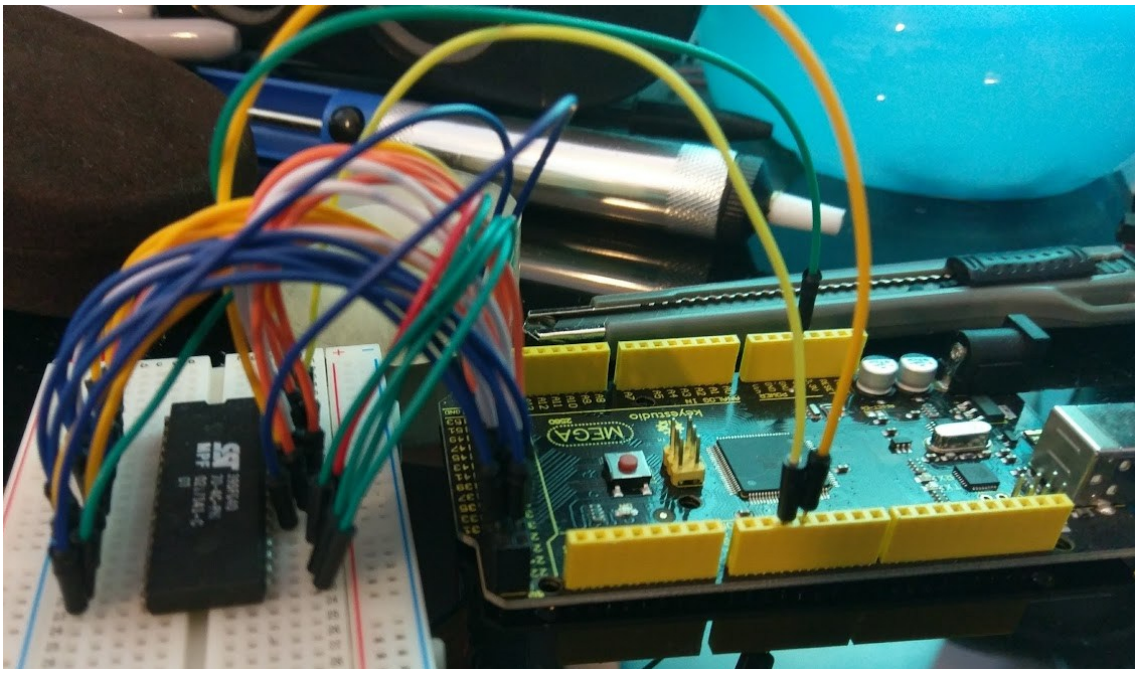
Notice the '#' after each of these pins, this means that they are active when the signal is driven low(0), for example, to enable DQ7 to DQ0 as output pins we must drive OE# low. Sometimes active low is shown by a solid line above the pin name instead of the '#' after the pin name.

## Choice of Arduino board

Initially, I was going to use an Arduino UNO for this project, but I encountered an issue. I would have to control the 30 pins on the flash chip with only 18 available pins on the UNO. This was possible by using some daisy-chained 8-bit shift registers to control the address pins and then the rest could be directly connected to the UNO. However, I soon decided that it would be easier, simpler and faster to use an Arduino MEGA and connect all of the pins directly to the MEGA.



(https://mint64.home.blog/imag0099/)



(https://mint64.home.blog/imag0605/)

## Reading from the device

Reading from the flash chip is the easiest and quickest of operations, to see how it's done we can look at the timing diagram provided in the data-sheet.

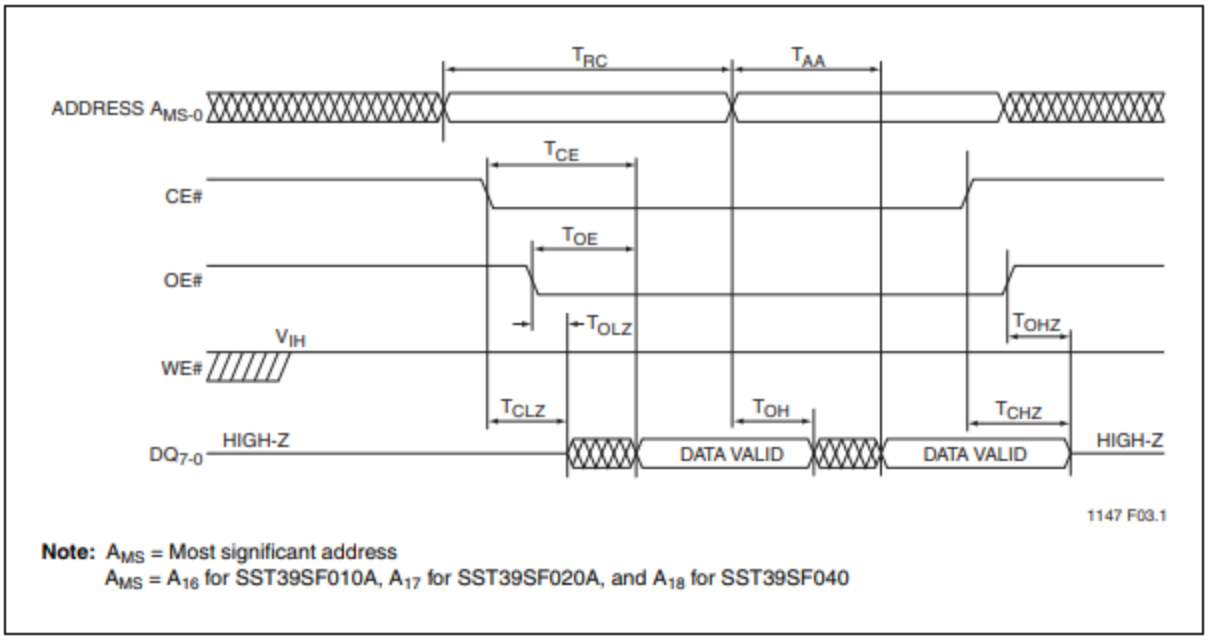



**Figure 5:** Read Cycle Timing Diagram

We can see that to read from the device we must have **CE# low, OE# low, WE# high** and the desired address present across A18 to A0. This will result in the stored byte being present across DQ7 to DQ0. Simple! Notice that there are lots of arrows with 'T' labels, these are the specific timing parameters for the different changes. We shouldn't have to worry about these too much right now.

## Writing to the device

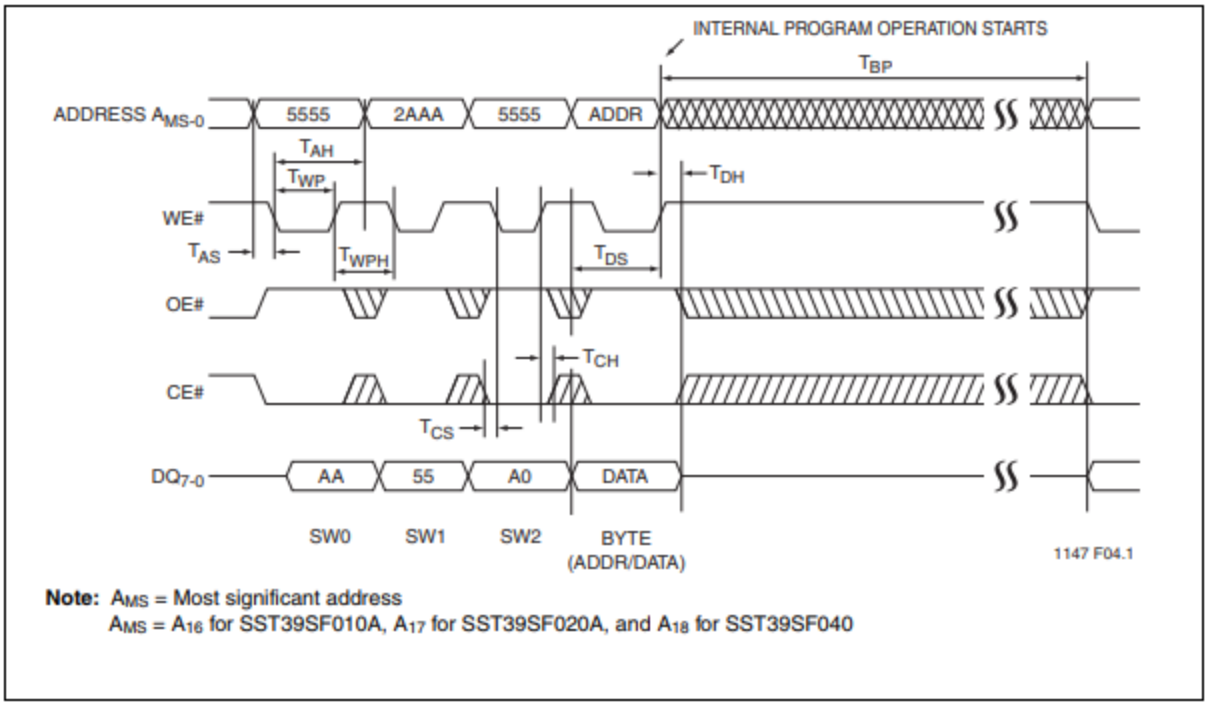
This is where things get slightly more complicated. In order to write to the device we can not simply drive WE# low, we instead have to send a specific command sequence to the device to unlock it and then we can write our data. To understand this we can first look at the "Software Command Sequence" table in the data-sheet.

**Table 4:** Software Command Sequence

| Command Sequence | 1st Bus Write Cycle | | 2nd Bus Write Cycle | | 3rd Bus Write Cycle | | 4th Bus Write Cycle | | 5th Bus Write Cycle | | 6th Bus Write Cycle | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Addr[1] | Data | Addr[1] | Data | Addr[1] | Data | Addr[1] | Data | Addr[1] | Data | Addr[1] | Data |
| Byte-Program | 5555H | AAH | 2AAAH | 55H | 5555H | A0H | $BA^2$ | Data | | | | |
| Sector-Erase | 5555H | AAH | 2AAAH | 55H | 5555H | 80H | 5555H | AAH | 2AAAH | 55H | $SA_X$[3] | 30H |
| Chip-Erase | 5555H | AAH | 2AAAH | 55H | 5555H | 80H | 5555H | AAH | 2AAAH | 55H | 5555H | 10H |

Firstly, we can see that the commands can be up to 6 write cycles long, where a write cycle is writing a specific byte to a specific address. All the information here is displayed in hexadecimal(hex), shown by the 'H' after all information, if you do not know what hex is I suggest you research and understand what it is. Basically, it can be used as an easy way to read and work with binary data as 4-bits can be shown as one digit 0 – F in hex.

Let's focus on the "Byte-Program" command sequence, its 4 cycles long meaning the first 3 are the commands to unlock and the last being our data and address. Now we should look at the timing diagram to see how to produce a write cycle.




**Note:** $A_{MS}$ = Most significant address
$A_{MS}$ = $A_{16}$ for SST39SF010A, $A_{17}$ for SST39SF020A, and $A_{18}$ for SST39SF040

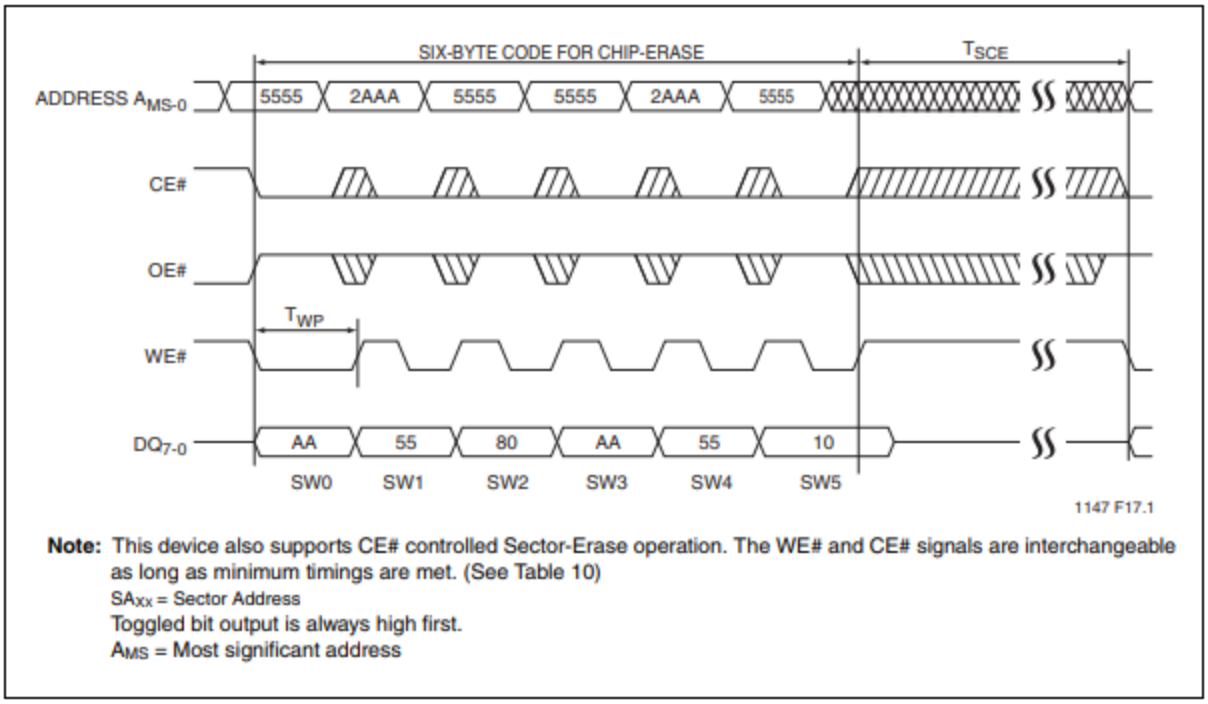
**Figure 6:** WE# Controlled Program Cycle Timing Diagram



We can see from this timing diagram that we need do drive **CE# low, OE# high** and then for each write cycle we **drive WE# low and then high between each cycle**. So what we want to do is set our address and data then drive WE# low, wait for a short time(1us), then drive WE# high and change the data and address then repeat the process.

## Erasing the device

Once again we must produce the correct command sequence to erase the chip, this will reset all the bytes to **FF** ( every bit is a 1 ). For EEPROMs like the AM29F040B, an erase should be done before the data is programmed as it can change 1s to 0s while writing but cannot change 0s to 1s, so all the data should be **FF**(1111 1111) before writing.



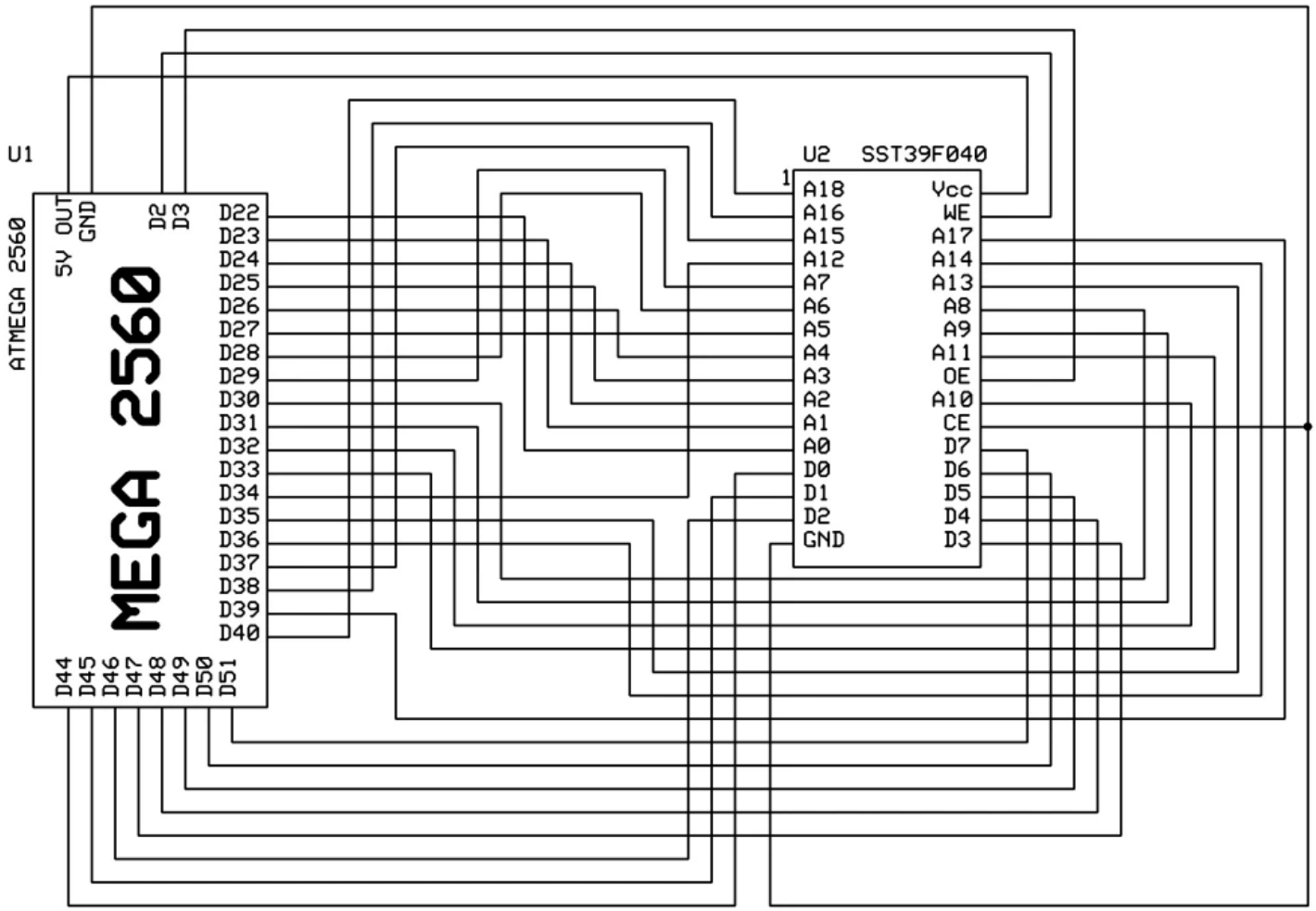

**Note:** This device also supports CE# controlled Sector-Erase operation. The WE# and CE# signals are interchangeable as long as minimum timings are met. (See Table 10)
$SA_{XX}$ = Sector Address
Toggled bit output is always high first.
$A_{MS}$ = Most significant address

**Figure 11:**WE# Controlled Chip-Erase Timing Diagram

Once again we must toggle WE# for each cycle and this should erase the flash chip after the 6 command write cycles are produced. There is also a sequence to erase particular sectors of memory but I am not going into detail on that.

# Connecting the MEGA to the SST39SF040

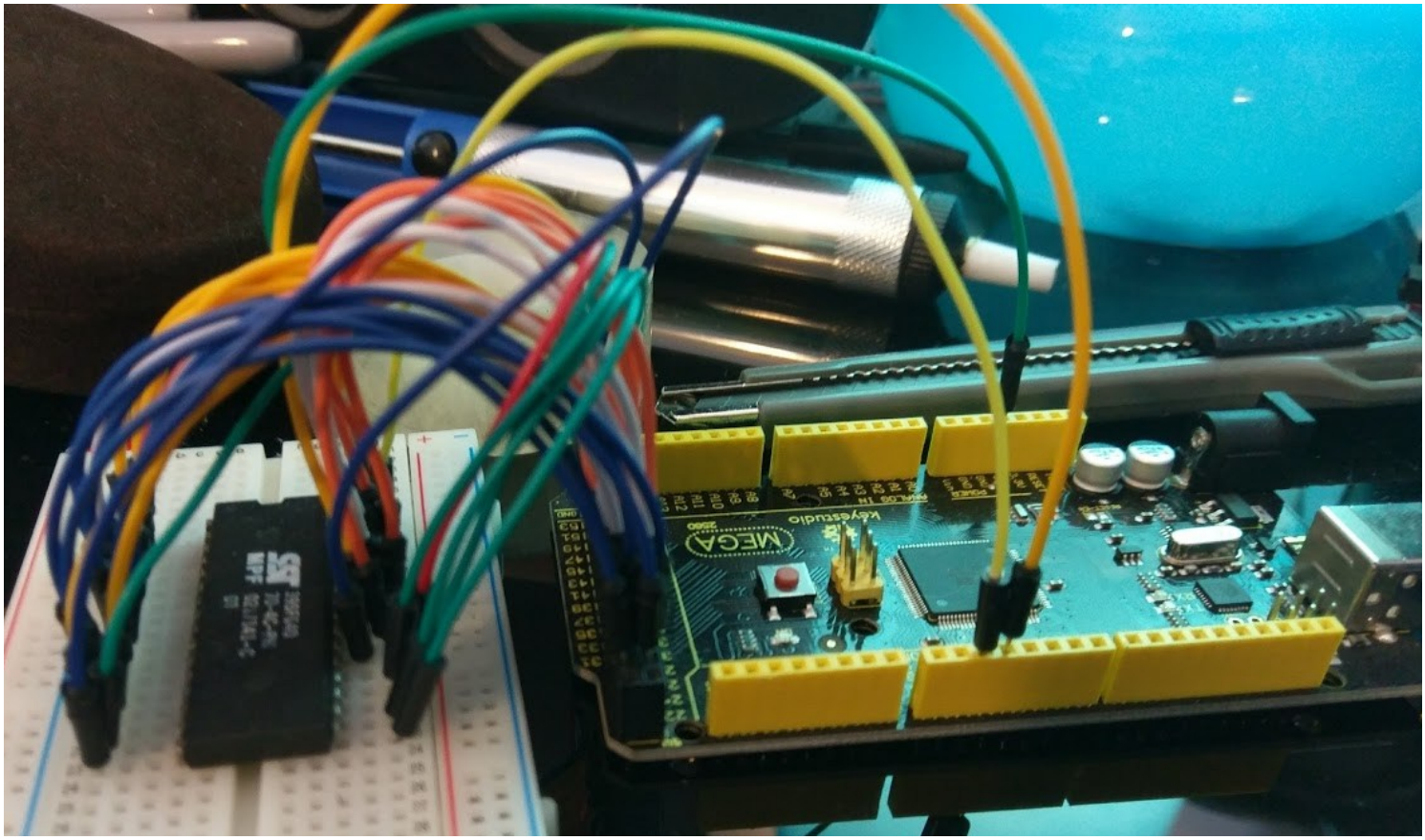
I simply chose various pins to connect to the Arduino and came up with this:




I apologise I have not exactly drawn this very well, it's pretty hard to read and see where each connection is going so I'll explain. I decided to start with the large header at the end of the MEGA and use the majority of its pins, I started at the first pin 22 and connected it to A0. Then I just connect each pin to the next, so 23 to A1, 24 to A2 and so on up to pin 40 which is connected to A18. Then I left a few out and started at pin 44 and did the same thing, connecting them to DQ0 to DQ7. The control pins are a bit different, I connected WE# to pin 2 and OE# to pin 3. I decided to just tie CE# straight to ground as in the operations it's always low. Finally, we just need to connect VDD to 5V and VSS to ground.

| SST39SF040 | ARDUINO MEGA |
|---|---|
| A0 - A18 | 22 - 40 |
| DQ0 - DQ7 | 44 - 51 |
| WE# | 2 |
| OE# | 3 |
| CE# | GND |
| VSS | GND |
| VDD | 5V |

This was all easily connected together on a breadboard as can be seen in this image:



So that's it for this part, now that we have an insight into how the flash chip works and connected it to our Arduino MEGA we should be ready to begin writing some code to see if we can get anything to work!

This is my first ever blog post and hopefully it's not too boring or poorly worded, next part I will write the code to follow what has been set out by this part.

:^)

Part 2: Arduino Code and Serial Comms (https://mint64.home.blog/2018/07/30/parallel-nor-flash-eeprom-programmer-using-an-arduino-part-2-arduino-code-and-serial-comms/)



**Tagged:**
am29f040b,
arduino,
eeprom,
nor flash,
programmer,
sst39sf040

# Published by mikemint64

*Hi! I'm a Northern Irish Electronic Engineering student currently studying in England. This blog is a place i can share any projects I work on in my spare time :) View all posts by mikemint64*