

SPI Flash- “The Big One”
Storing Persistent Data on an M95P32 SPI Flash Chip

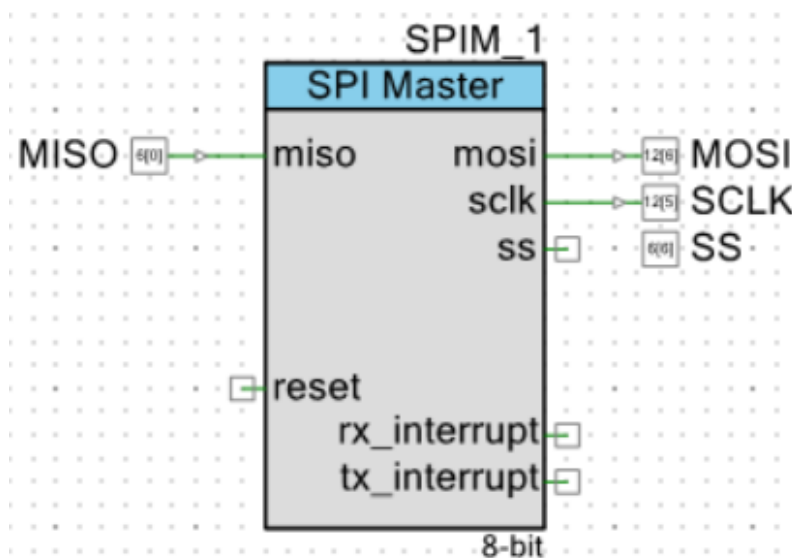
Have music clips in your project? Want to keep your diary in a PSoC typewriter?

Consider using an M95P32 Mbit SPI flash memory chip! These chips connect to your PSoC over a speedy serial interface (SPI, Serial Peripheral Interface), save memory space on your PSoC, and don’t get reset on power-down. Sounds like an SD card, right? Well, the advantage of these chips is that, unlike SD cards, you can read, write, and over-write individual bytes! No more 512-bit blocks, no more read-modify write. This 3.3 V 32 Mbit can read/write at about 10 kbps with the demo code, and can be clocked up to about 35 kbps without changing much. Theoretically, you can get up to around 5 MBps, but that takes much more effort to implement.

Before we dive into it, though, let’s talk about a couple of limitations of the M95P32. First, it’s a surface-mount part, so you’ll need a “surf-board” to adapt it to your bread board. (If you really need flash in a DIP package, check out the SST25VF0808B chip on the PSoC Information page) Second, this chip runs at 3.3 V, meaning you’ll need to jump through some extra hoops to use it with the PSoC Stick instead of the Big Board. (For a 5 V flash chip, check out the M32M04 instead!)

This guide serves as a sort of quick-start guide, and as such doesn’t have a lengthy explanation of the SPI protocol. For a more in-depth explanation of SPI on the PSoC, please see the M95P32 Full Guide on the PSoC Information page.

Locking away your precious data may seem daunting, but the interface is simpler than it sounds. At the core is this block: the SPI Master. (also now commonly referred to as the SPI Controller, but for consistency with the PSoC documentation we’ll be using the original terminology)



The block has four main connections:

MISO: Master In Slave Out (Controller In Peripheral Out)

Data line that moves data out of the Master and into the Slave. Set this pin to resistive pull-up.

MOSI: Master Out Slave In (Controller Out Peripheral In)

Data line that moves data out of the Slave and into the Master. Set this pin to strong drive.

SCLK: SPI Clock

This line controls the timing of data transfers. Set this pin to strong drive.

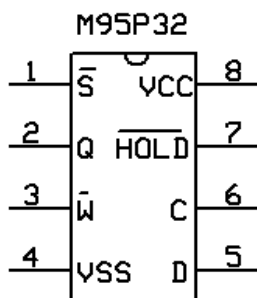
SS: Slave Select (Chip Select)

This line activates a given chip on the SPI bus. Active low (pull down to select chip)
Set this pin to strong drive.

Notice that, for this project, we have our own SS pin, not connected to the SPI Master. This is necessary due to the way the SPI Master handles the SS line.

Hooking up to the flash is as easy as directly connecting to the corresponding pins on the chip (**IF AND ONLY IF YOU ARE USING THE BIG BOARD IN 3.3 V MODE. ADDITIONAL CONSIDERATIONS ARE NECESSARY TO INTERFACE USING 5 V**)

The pins on the chip are:



!S: Select. Active low: pull low to select the chip

Q: Data out. (MISO)

!W: Write protect. Active low. When low, it locks the write-protect bits.

Vss: gnd of the chip

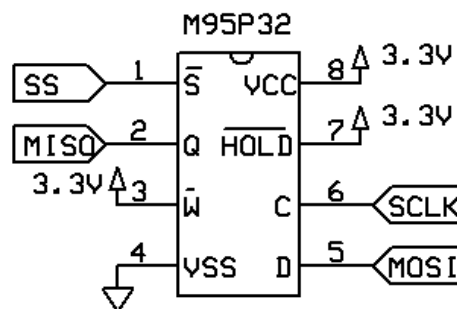
D: Data in. (MOSI)

C: Clock (SCLK)

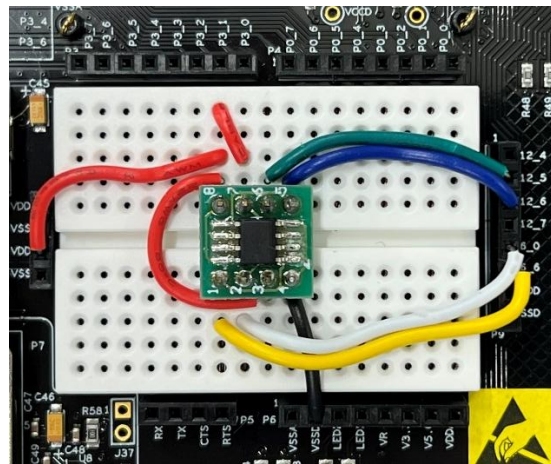
!HOLD: Active low. When activated, pauses the chip

Vcc: power supply for the chip

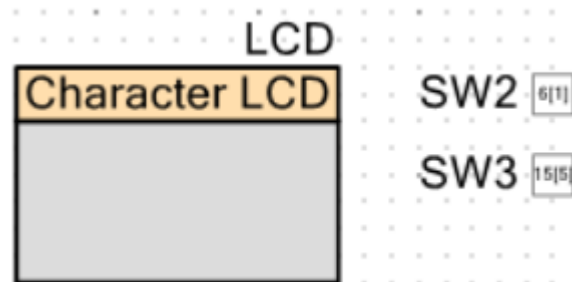
We will connect these pins as follows:



And here it is on the big board:



We also have a character LCD and two buttons for the demo, and a char LCD block. Both button pins should be set to 'resistive pull-up.'



The pins should be assigned as shown in the Design Wide Resources (.cydwr):

| | Name | Port | Pin | Lock |
|-------------------------------------|--------------------|---------|-------------|-------------------------------------|
| <input checked="" type="checkbox"/> | \LCD:LCDPort[6:0]\ | P2[6:0] | 2,1,99...95 | <input checked="" type="checkbox"/> |
| <input checked="" type="checkbox"/> | MISO | P6[0] | 89 | <input checked="" type="checkbox"/> |
| <input checked="" type="checkbox"/> | MOSI | P12[6] | 29 | <input checked="" type="checkbox"/> |
| <input checked="" type="checkbox"/> | SCLK | P12[5] | 5 | <input checked="" type="checkbox"/> |
| <input checked="" type="checkbox"/> | SS | P6[6] | 8 | <input checked="" type="checkbox"/> |
| <input checked="" type="checkbox"/> | SW2 | P6[1] | 90 | <input checked="" type="checkbox"/> |
| <input checked="" type="checkbox"/> | SW3 | P15[5] | 94 | <input checked="" type="checkbox"/> |

Now, for the demo!

- Go ahead and wire up the chip to the big board, install the character LCD, and flash the “M95P32 SPI Flash Demo” project onto your PSoC. (Or copy the code below into main.c)
 - With the PSoC Powered on, press SW2 to save parts of the string “6.115 is cool!!” to the flash chip.
 - The first press will save “6.115”, then the second will add on “ is”, and the third will save “ cool!!” Presses after the third do nothing.
- After saving some portion of the string to the flash chip, press SW3 to read the string back out and print it to the LCD.
 - This can be done with any portion of the string saved, not just the whole thing!
- Also be sure to unplug the PSoC after saving some part of the string, and then press SW3 without any presses on SW2.
 - Notice the chip still remembers what was on it! This is the magic of flash memory.

Cool! Now let’s go over the code and how it works:

The first thing we need to do to get the chip working is disable write protection. This is done by selecting the chip and sending the “Write Enable” (WREN) command, hex code 0x06, to the chip. This is required before every write instruction, to tell the chip to accept the input. We need to raise and then lower the select line to ‘latch’ the command, and start the next one. Then, we send two bytes: 0x01 (Write status register or WRSR command code) and then 0x00, writing all block protection bits to zero. Raising the select line finishes the command.

To write to the chip, as seen inside the if statement, we first send the WREN command as we did before, and then we lower the select line and send the byte 0x02, for WRITE. Then, we transmit the 24-bit address one byte at a time, with the most significant byte first. After this, we transmit as many bytes as we’d like to write. The chip will keep accepting bytes, automatically incrementing the address we write to. (However, once we’ve crossed a “page boundary”, at multiples of 512 bytes, it starts again on the same page, so you’ll need to be aware of this for larger writes). After all bytes are transmitted, we once again raise the select line.

The last thing we need to be able to do is read. To read, we lower the select line and send byte 0x03, the READ command. Then, we send the 24-bit address we want to begin reading from, as we did with the WRITE command. After that, we write as many dummy bytes as bytes we want to read, since the SCLK line only runs when the SPI Master is transmitting. These bytes are ignored by the chip, which sends the data we want back on the MISO line. We can read these bytes in from the SPI receive buffer, and then print them to the screen.

For simple reads and writes, that’s all there is to it!