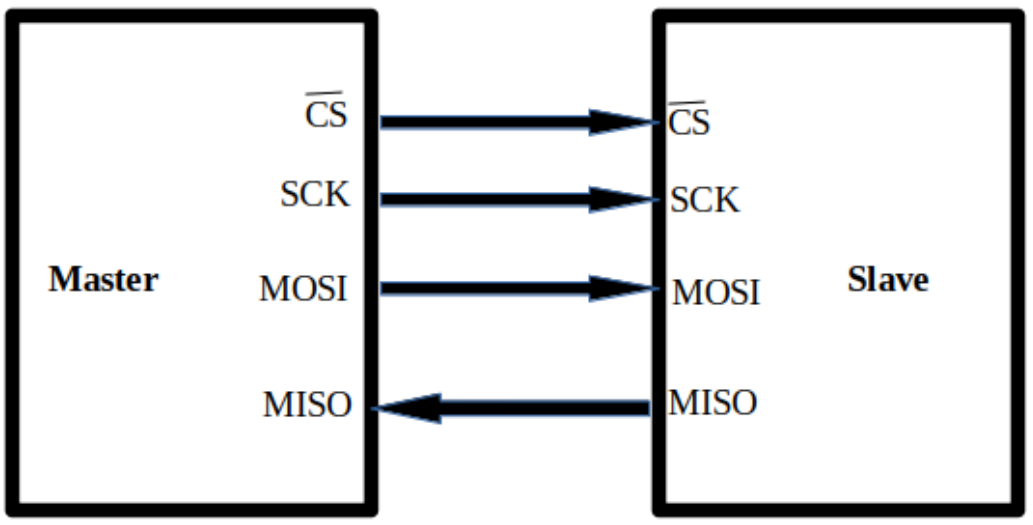
IoT Security - Part 15 (101 - Hardware Attack Surface : SPI)

Asmita-Jha

26-September-2020



This blog is part of the IoT Security series, where we discuss the basic concepts about the IoT/IIoT eco-system and its security. If you have not gone through the previous blogs in the series, I will urge you to go through those first. In case you are only interested in hardware SPI, feel free to continue.

IoT Security - Part 1 (101 - IoT Introduction And Architecture)

IoT Security-Part 14 (Introduction to and Identification of Hardware Debug Ports)

Introduction

An IoT product eco-system comprises a hardware device(s), firmware, network, communication protocols, web, mobile, cloud applications, and the eco-system is growing exponentially with growth in the IoT market. It comprises of a wide range of attack surfaces. To get a better understanding of the IoT attack surface, you can refer IoT Security - Part 2 (101 - IoT Attack Surface). An embedded device has many communication protocols at the hardware level that enable the microprocessor/microcontroller to communicate with its peripheral components.

This blog will discuss one of the hardware communication protocols, **Serial Peripheral Interface (SPI)**. If you are a beginner at hardware hacking and want to get started at a basic level, stay tuned, you are in the right place.

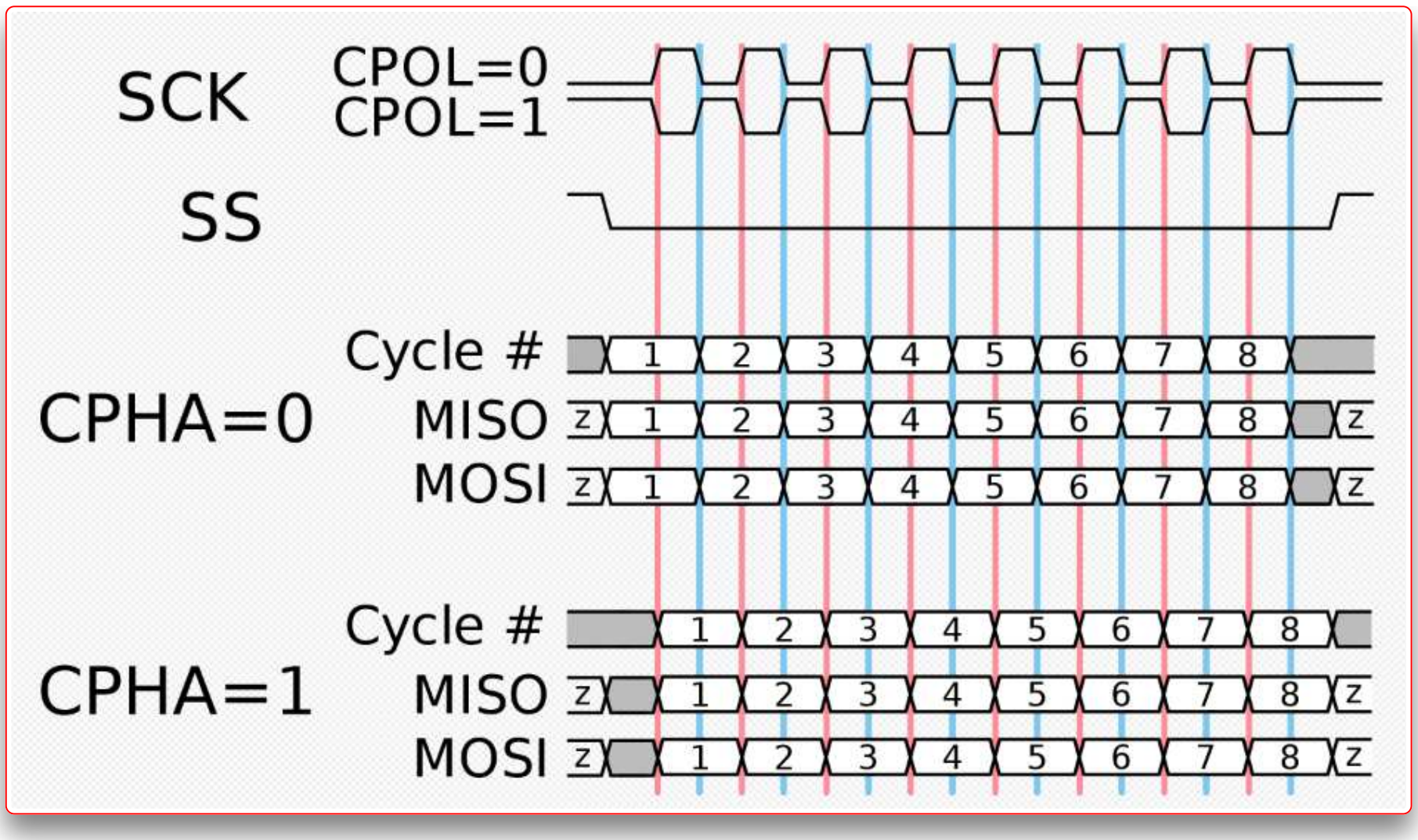
SPI is a synchronous serial communication interface. It was designed primarily to communicate (transfer data) between the components located on the same PCB (Printed Circuit Board). It was developed by Motorola. It provides full-duplex communication mode (i.e., sender and receiver can transmit and receive the data simultaneously). It has a master-slave architecture that supports a single master and multiple slaves. It is a 4-wire serial interface. It consists of the following four logic signals :

- MOSI (Master Output -> Slave Input) - Master outputs the data to the slave.
  - SDO, DO, DOUT, SO - These are different names for MOSI pins on the master device that connects to MOSI pins on the slave device.
  - SDI, DI, DIN, SI - These are different names for MOSI pins on the slave device that connects to MOSI pins on the master device.
- MISO (Master Input <- Slave Output) - Slave outputs the data to the master.
  - SDO, DO, DOUT, SO - These are different names for MISO pins on the slave device that connects to MISO pins on the master device.
  - SDI, DI, DIN, SI - These are different names for MISO pins on the master device that connects to MISO pins on the slave device.
- SS (Slave Select/Chip Select) - Output from master. Master uses this signal to select the slave device.
- SCLK (Serial Clock) - Output from the master. To start the communication with the slave, the master configures and sends the clock signal. It also provides the speed of data transfer between the master and the slave.

SPI supports 4 basic modes of operation :

SPI Modes	Clock Polarity (CPOL/CKP)	Clock Phase (CPHA/CKE)
Mode 0	0	0
Mode 1	0	1
Mode 2	1	0
Mode 3	1	1

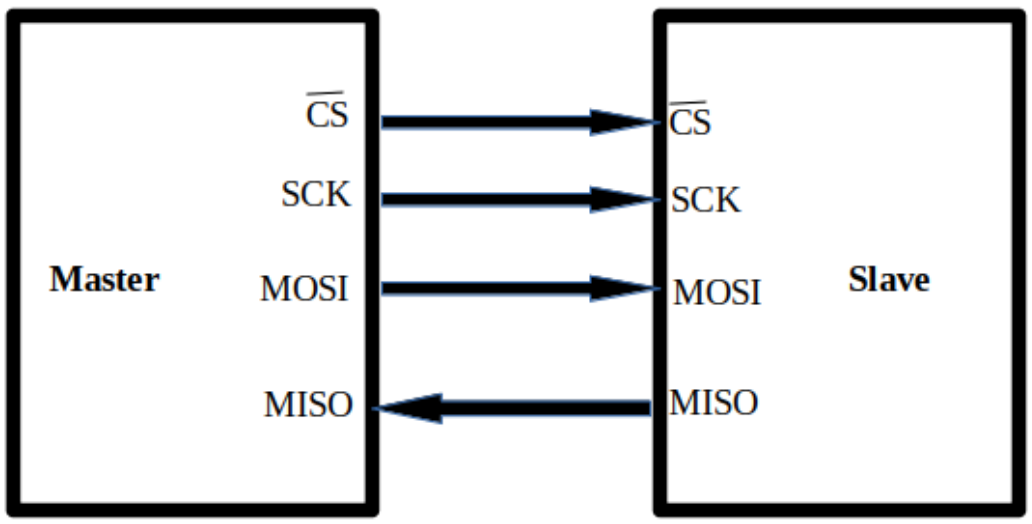
All the SPI devices do not support all these modes. Different SPI devices may either work on logic 0 or logic 1. They may have different data latching mechanism. For example, they may support latching data either at rising or falling edge; some may support either low or high as an idle state. So, having these different SPI modes provides the flexibility to different types of devices to perform SPI communication. The mode supported by any SPI device can be found in their respective datasheet. These modes operate similarly but have different timing relation with the SCK clock edge. The timing diagram showing different modes with clock polarity (CPOL/CKP) and clock phase(CPHA/CKE) is shown below. (Source - wikipedia page)



Depending on the mode that is to be used for communication, the master configures clock polarity and clock phase.

- If CPOL/CKP = 0 : Clock is idle at 0. The leading edge is a rising edge, and the trailing edge is falling.
- If CPOL/CKP = 1 : Clock is idle at 1. The leading edge is falling, and the trailing edge is a rising edge.
- If CPHA/CKE = 0: SDO shifts the data on the trailing edge of the preceding clock cycle, and SDI samples the data on the rising edge.
- If CPHA/CKE = 1: SDO shifts the data on the leading edge of the current clock cycle and SDI samples the data on the falling edge.

Moreover, the combination of CPOL and CPHA gives the different modes of SPI communication. The diagram below depicts the high-level SPI communication between the master and the slave.



1. Master selects the slave device by sending logic 0 on the CS signal. (CS is active low)
2. Master sends the clock signal
3. Master sends the data/command on the MOSI line, and the slave reads it
4. Slave sends the data/response on the MISO line, and the master reads it

There are two shift registers involved, of given word size, often 8 bits, although other word sizes are also common. One shift register is present in master, and another is present in the slave. They form an inter-chip circular buffer. The most significant byte (MSB) of the data is usually shifted out first. Its detailed explanation is available at wikipedia page.

Possible Attack Scenarios

In an embedded device, the SPI protocol is used to communicate with various peripherals on the device. It could be sensors, control devices, LCD, SD Card, EEPROM, Flash memory, etc. If an attacker gets access to the device hardware, there are possibilities that he may find the SPI device as an attack surface on the hardware.

- Sniffing the communication between an SPI device like sensors, controlling device, memory chip, etc. and the controller/processor. It may lead to the stealing/leak of sensitive information by the attacker. For example, in some of the embedded devices, an SPI EEPROM is used to store sensitive information, keys, logs, etc. This sensitive info can be leaked if an attacker sniffs the SPI communication between the EEPROM and the controller/processor. After getting this sensitive info, an attacker can damage in many ways and maybe on a large scale if the hardware vendor has used the same sensitive info like the same keys in all the hardware in the production.
- Extracting the firmware/sensitive info from SPI memory chips. Some embedded devices use SPI Flash memory to store firmware inside it. There are already tools/ methods available to read NAND/NOR SPI/Parallel flash memories. The attacker can read the flash memory and extract the firmware out of it. The firmware can further be reverse engineered and analyzed for potential vulnerabilities or cloning. If the firmware is not protected, the IP can be easily

stolen. If the chips hold configuration or other sensitive data (instead of firmware), extracting and analyzing can lead to unearthing new vulnerabilities, as suggested in the above point.

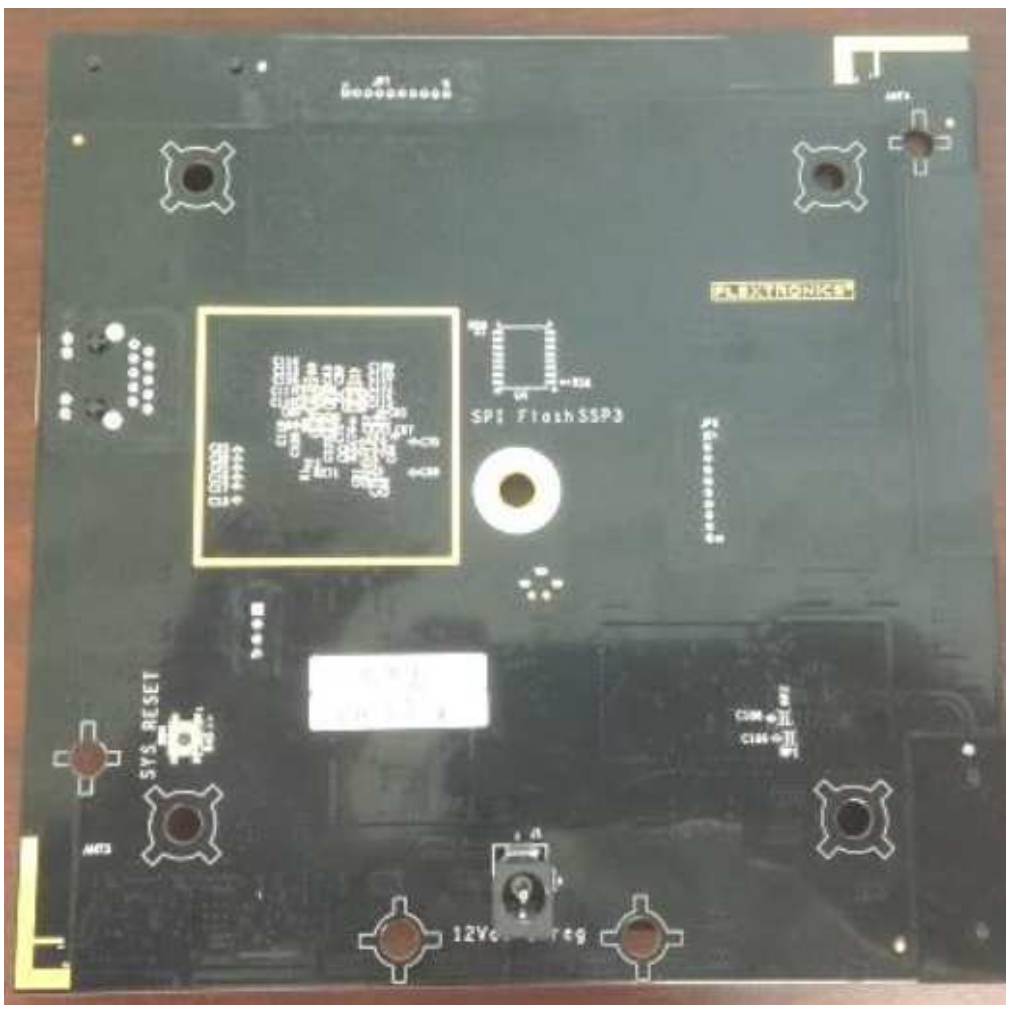
- Patching the firmware in the SPI flash memory chip. In this case, if the firmware is not securely managed on the device, an attacker can patch the firmware with a backdoor and reprogram the SPI flash memory with the malicious firmware.

Attacking hardware via SPI

There are various ways in which the attacker can attack the hardware via SPI. Few methods to perform the attacks are explained in the below section.

Recon

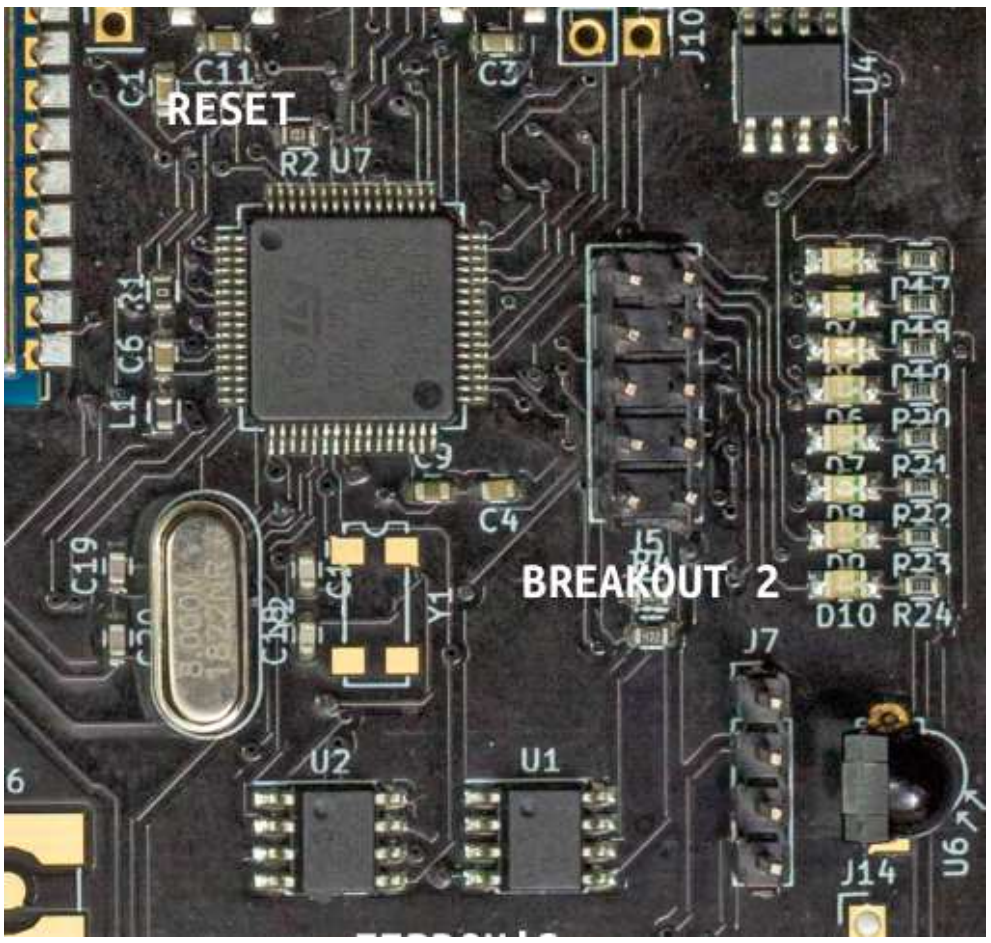
Case 1: You do not have the actual hardware with, but you know, the FCCID number of the device. Go to the FCCID website, search for the FCCID number of the device. If correct, you will find all the internal images and detailed internal, external information about the device. Seeing the internal image, you may get the hint for an SPI chip on the hardware. Yayy!! Once you know that you got the attack surface, you can get/purchase that device and perform further required steps to attack the device. For example, We looked for a Wink hub device, we were able to find the FCCID number of the device and searched for it. We found its detailed internal image here. It was found that the device contains an SPI Flash, as shown in the image below.



Voila!! We found an SPI chip on the device. Let us go, fetch & attack :)

Case 2: You got access to the hardware. Once the hardware of a device is found, the first step should be to perform reconnaissance. Inspect each chip, test points present on the printed circuit board (PCB). You can read more about hardware recon here.

- If successful in identifying the SPI chip used on the device, search for its datasheet on the Internet, identify the pin diagrams given, and trace those pins on the PCB. Check the continuity between the controller's SPI pins and the test points present on the PCB using **Multimeter**. For example, We are showing here the image of **EXPLIoT DIVA board**.



It contains some chips on the PCB. We googled the datasheet for each chip and found that one of its chips denoted as U2 is an **SPI EEPROM 25LC256** After finding the chip, the next task is to identify that if the SPI pins (MISO, MOSI, SCK, CS) of the chip is connected to some points on the PCB or not. Fetching the datasheet of 25LC256, we identified the respective pins of the chip as shown in the image below (Source - 25LC256 datasheet)

- Block Write Protection:
  - Protect none, 1/4, 1/2 or all of array
- Built-In Write Protection:
  - Power-on/off data protection circuitry
  - Write enable latch
  - Write-protect pin
- Sequential Read
- High Reliability:
  - Endurance: 1,000,000 erase/write cycles
  - Data retention: >200 years
  - ESD protection: >4000V
- Temperature Ranges Supported:
  - Extended (M): -55°C to +125°C
- RoHS Compliant

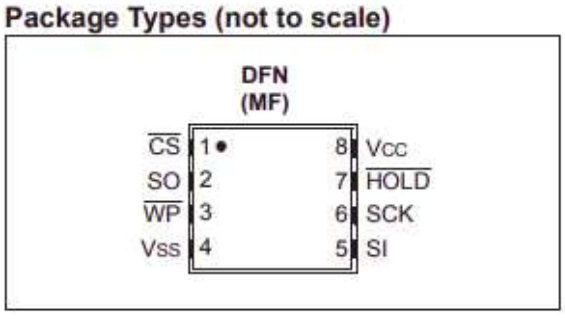
Pin Function Table

Name	Function
$\overline{CS}$	Chip Select Input
SO	Serial Data Output
$\overline{WP}$	Write-Protect
Vss	Ground
SI	Serial Data Input
SCK	Serial Clock Input
$\overline{HOLD}$	Hold Input
Vcc	Supply Voltage

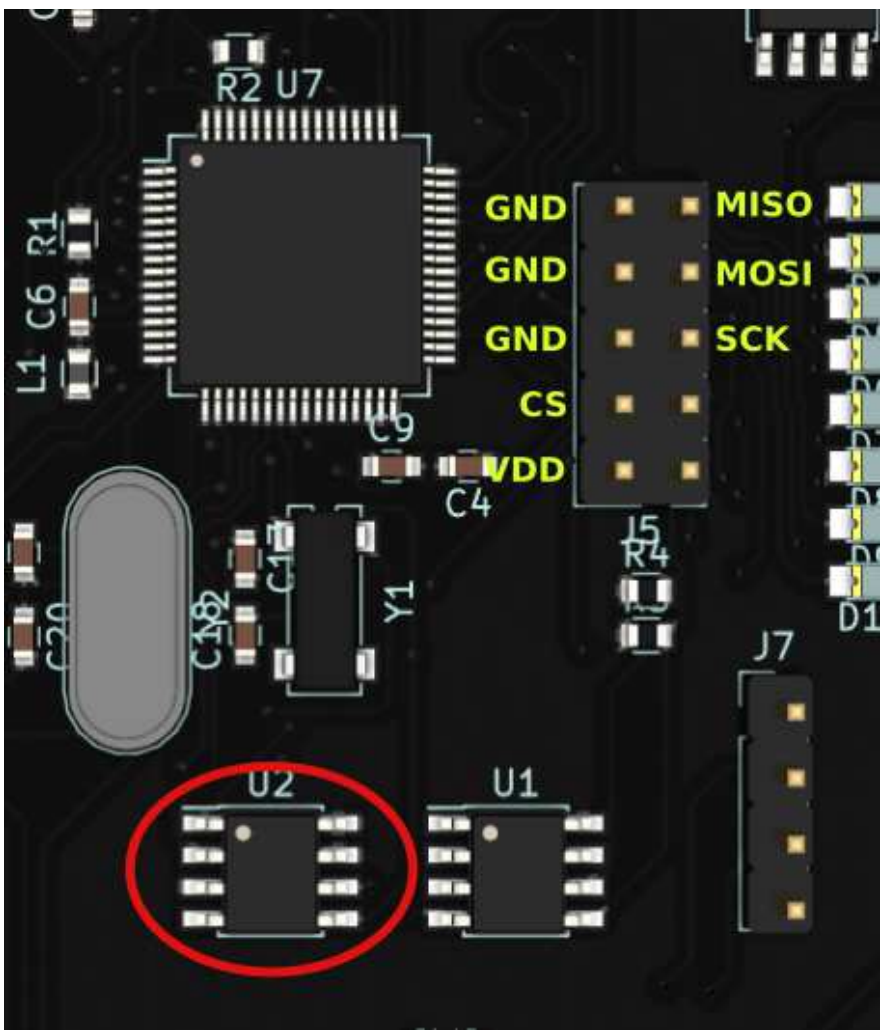
exception of Chip Select, allowing the host to service higher priority interrupts.

The 25LC256 is available in the 8-lead DFN package.

Package Types (not to scale)



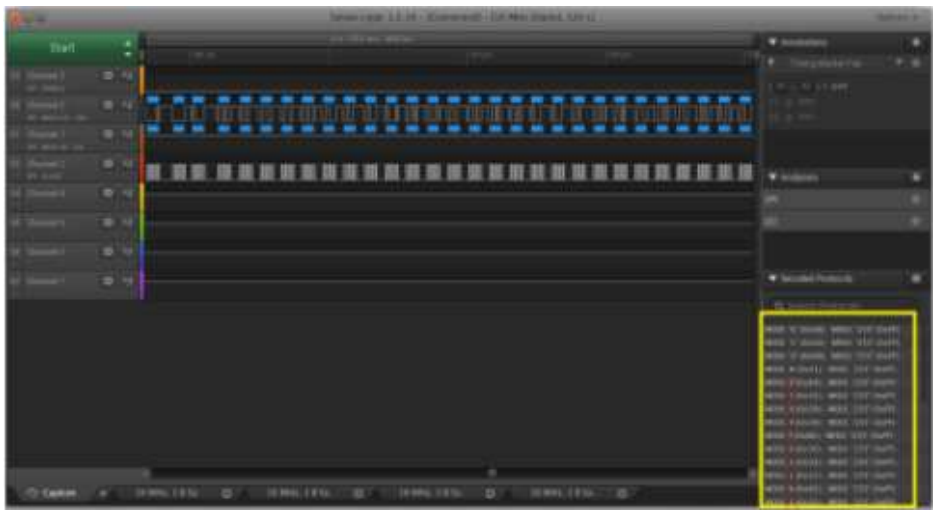
Using the multimeter, we checked the connectivity of these SPI pins with other pins on the PCB. We found that pins on the J5 header, as in the image below were connected to these SPI pins.



Yayy!! We can now use these pins to sniff, extract and attack the device :)

Sniffing SPI Communication

Now, once the SPI pins on the hardware are successfully identified, we can go ahead and sniff the communication between the SPI chip and the controller. Tools like [Logic Analyzer] can be used to sniff the communication on the SPI bus.



The logic analyzer shows the signals on all the four SPI pins i.e., MISO. MOSI, CLK, CS. Decoding these signals as per the protocol timing diagram. Softwares like Saleae Logic Analyzer, PulseView have the feature to detect these protocols that directly shows you the decoded data w.r.t the signals. Yayy!! If the data transfer is in plain text, we can sniff and get it.

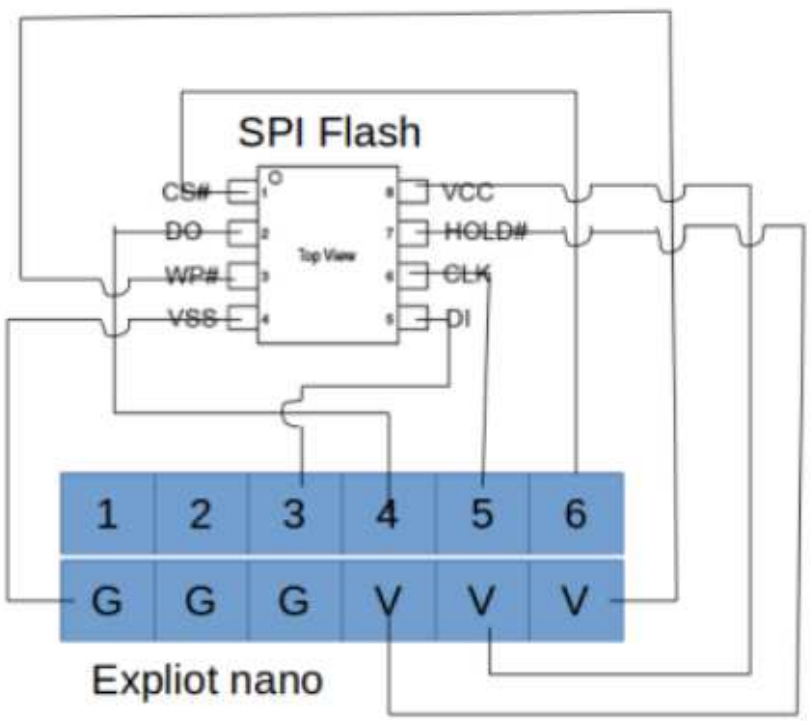
Extracting data/firmware from SPI Memory Chip.

To communicate with the SPI chip present on the hardware, we need protocol adapter tools that support SPI protocol and framework to make it feasible to communicate those chips without the host machine. (Sometimes, we also need to desolder the flash memory chips from the hardware, solder it on perf board or put it in suitable TSOP socket and connect the requited pin to the adapter.) There are various tools and frameworks available to extract the data from the memory chips. A few of them are mentioned below.

1. **EXPLIoT Nano**
2. **Bus Pirate**
3. **Shikra**
4. **CH341A**
5. **EXPLIoT Framework**
6. **Flashrom**
7. **pyspiflash**
8. **pyftdi**
9. RaspberryPi (refer **here**) or Beaglebone (refer **here**) can also be used.

Depending on the kind of memory chip that we have like EEPROM or FLASH and corresponding support by any of these devices, we can select a set of tools to extract the date/firmware. First, we need to make sure whether the SPI chip that we have is supported by these tools. If it is supported and **voltage level** matches, we proceed with that tool or framework. In case the voltage level of the adapter tool and the SPI chip do not match, use level shifters.

Case 1: You are successful in finding the tools or framework that can support your SPI memory chip Let us say we find an SPI Flash memory in a camera's hardware, and we want to extract the firmware from the chip. Our approach would be as mentioned below. 1. Take out the datasheet of the SPI flash memory. Identify the pins and voltage required. 1. Accordingly, choose the adapter tool that can be used. Connect the pins of the SPI flash memory to the suitable adapter, for example, EXPLIoT Nano as per its datasheet/pinout manual. The connection image is shown below.



1. Then, in your host machine, install the framework supported by the adapter that has been used. For example, **EXPLIoT Nano** works with **EXPLIoT Framework**, we set up this framework and use its plugin run spi.generic.readflash -w camera_dump.bin to extract the firmware from the flash memory. An example image is shown below.

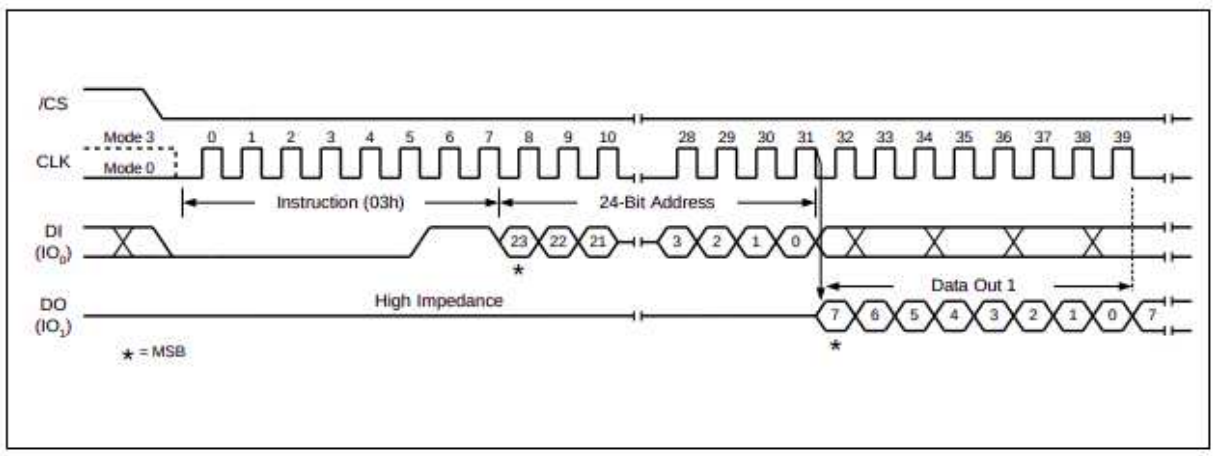
```
ef> run spi.generic.readflash
[*] Test:         readflash
[*] Author:       Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://github.com/eblot/pyspiflash']
[*] Category:     Protocol=spi|Interface=hardware|Action=analysis
[*] Target:       Name=generic|Version=generic|Vendor=generic
[*]
[*] Reading data from spi flash at address(0) using device(ftdi:///1)
[+] (chip found=Winbond W25Q64 8 MiB)(chip size=8388608 bytes)(using frequency=30000000)
[?] Reading 8388608 bytes from start address 0
```

Similarly, other tools and frameworks as suited can be used to dump the data/firmware from the SPI memory chip.

Case 2: You cannot find any tool/framework that can support your SPI memory chip In this case, the hardware setup steps would be the same as in the above. Take any FTDI232H based adapter, match the voltage levels of the SPI chip and the adapter. Do connection as per adapter and the chip's user manual/datasheet.

In this case, you need to write your framework, or you can even add the currently unsupported flash chips to any of the open-source frameworks of your choice. So, for this, you first need to read the datasheet of the SPI memory chip carefully. The datasheet contains all the details required to write your script to read/write the data from/to the memory chip. For example, SPI Flash memory W25Q256JV, let us have a glance at its datasheet. It gives the detailed instructions and commands required to read, write, and perform other actions. The image below from W25Q256JV datasheet, shows the read data instruction diagram.

The Read Data (03h) instruction is only supported in Standard SPI mode.



Similarly, the below image shows SPI write instruction.

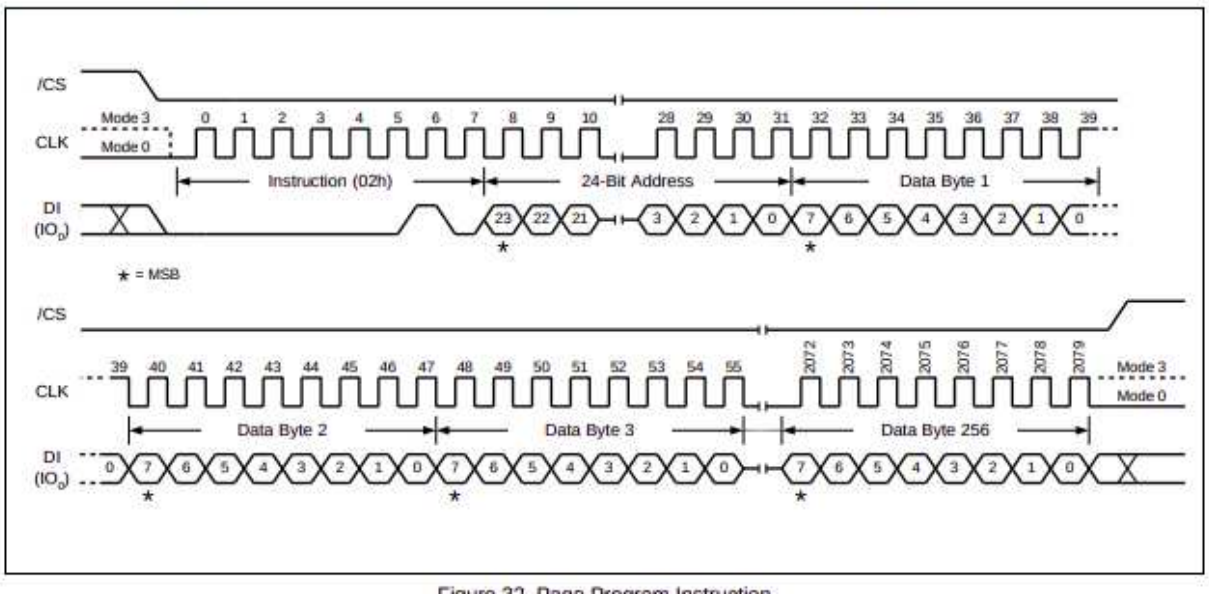


Figure 22. Page Program Instruction

So, following the SPI memory chip datasheet instructions, one can write a script to dump or program the chips. It can also be done using Raspberry Pi or beaglebone. We will post detailed SPI memory extraction via real-world example scenarios, different kinds of issues we face while dealing with specific SPI chips, and how to solve those issues in future blogs. So, stay tuned :)

Measures that can mitigate the attacks

- Encrypt data on the memory chip to prevent leakage of sensitive info.
- Do not hardcode sensitive data on the memory chips.
- Store encrypted firmware/data
- Physical hardening and protection of chips There are also a few crypto-based memory chips like **ATAES132A** and some other chips as given on **Microchip website**. Their datasheets claim to provide authentication and encryption features; secure read/write access. Using crypto-based memory chips can increase the complexity of the attacker to steal the data.

Conclusion

In this blog, we learned about the SPI protocol, its application, the possibilities of attack scenarios, the methods to attack, and few preventive measures. We hope you enjoyed and got some valuable information out of it. These tools and attack methods would give you a basic understanding of what to do when finding an SPI chip on the device hardware. With that, we hope, pretty soon, you get to say, "Voila!! I found an attack surface in the hardware, let's attack".

Continue to next Part - IoT Security - Part 16 (101 - Hardware Attack Surface: I2C)