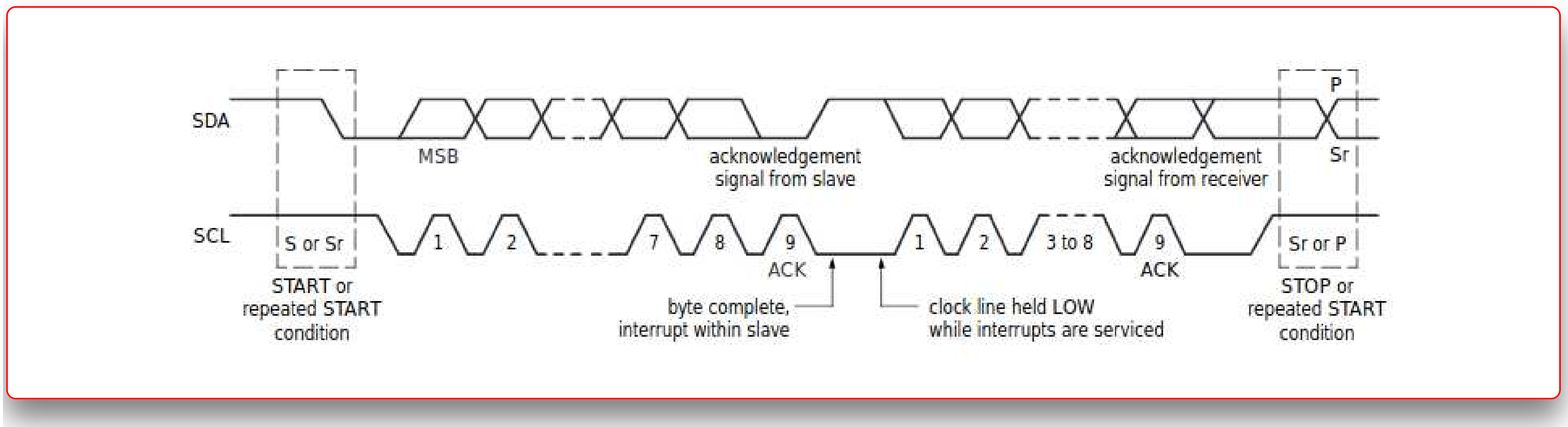
IoT Security - Part 16 (101 - Hardware Attack Surface: I2C)

Asmita-Jha

27-September-2020


SDA
SCL
MSB
acknowledgement signal from slave
acknowledgement signal from receiver
P
Sr
S or Sr
1
2
7
8
9
ACK
3 to 8
Sr or P
START or repeated START condition
byte complete, interrupt within slave
clock line held LOW while interrupts are serviced
STOP or repeated START condition

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 I2C, feel free to continue.

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

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

In the previous part of this blog series, we had discussed the SPI protocol. This blog will discuss another hardware communication protocol, **Inter-Integrated Circuit (I2C)**. So, if you are a beginner in hardware hacking and want to get started with basics, stay tuned; you are at the right place.

Introduction

I2C is a synchronous serial communication interface. It is primarily used for short-distance intra-board communication. It was developed by Philips Semiconductor(now NXP Semiconductors). It provides a half-duplex communication mode (i.e., sender and receiver can transmit/receive the data one at a time, not simultaneously). It has a master-slave architecture that supports multiple masters and multiple slaves. It is a 2-wire serial interface.

It consists of two bidirectional open-collector(i.e., the device can pull the respective lines low but cannot drive it high. Hence, this avoids any contention between the devices when one is trying to drive the line high, and the other is trying to pull it low.), lines pulled up with resistors (it keeps the line high when the line is not pulled low by any device) :

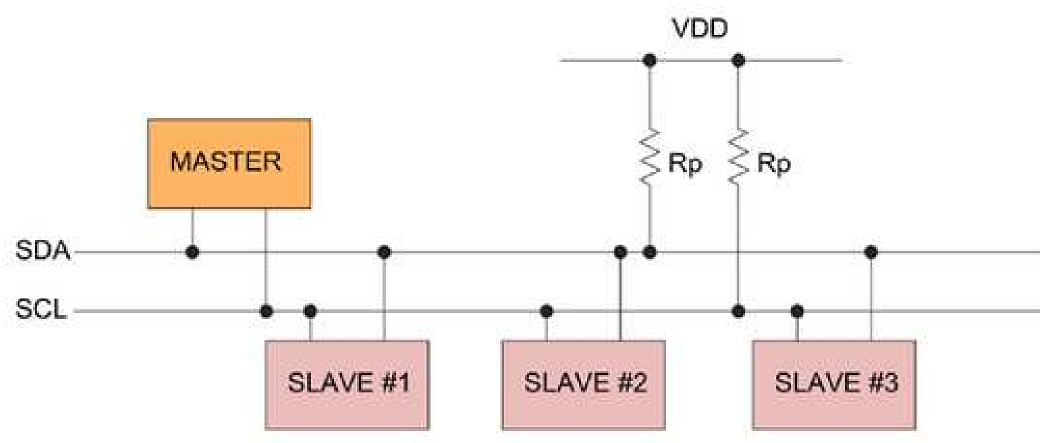
- Serial Clock (SCL)
- Serial Data (SDA)

It supports different bidirectional speed modes 100 kbit/s standard mode, 400 kbit/s fast mode, 1 Mbit/s fast mode plus, 3.4 Mbit/s high-speed mode, and unidirectional 5 Mbit/s ultra fast-mode (these bit rates are without any clock stretching or any other overhead). Multiple masters can exist simultaneously on the same bus. The I2C bus has two nodes :

- Master node - This node generates the clock and starts the communication with slaves.
- Slave node - It receives the clock and communicates with the master if addressed by it.

Each device connected to the I2C bus has a unique address. The number of nodes on the bus is determined by the address space and the total capacitance of the bus, i.e., 400 pF (Source - wikipedia). It has a 7-bit or 10-bit address space. Further on, we will be discussing w.r.t 7-bit address space. In 7-bit, the maximum number of devices that can be connected over I2C bus is 128. Apart from data bits, I2C has START and STOP bits that indicate the beginning and end of the communication between the master and the slave.

The image below shows the general connection diagram in I2C (Source - here).



1. Master initiates the communication by sending the START bit (SDA line is pulled low when SCL is high, then the SCL is pulled low) followed by the 7-bit/10-bit address of the slave it wants to communicate with, it is followed by 1-bit indicating if it wants to write to (0) or read from (1) the slave.

2. If the slave exists, it responds to the master with an acknowledgment bit ACK.

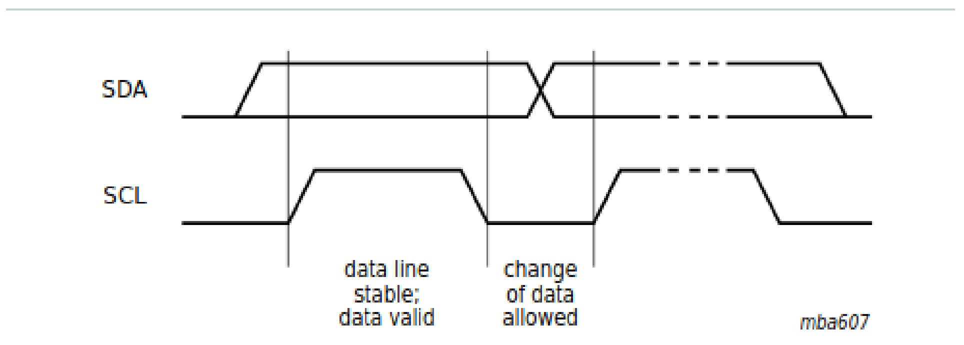
3. Then, depending on the read or write operation, the master continues to either transmit or receive, and the slave operates in the complementary way, i.e. receive or the transmit, respectively. The size of the data frame that is transmitted is 8-bit. The most significant bit (MSB) of the data byte is transferred first. While transferring, the data is put on the SDA line after SCL is pulled low. During the high period of SCLK, data on the SDA line should be stable. The HIGH or LOW state the SDA line changes only when the SCL line is LOW.

   - In case of the write operation, the master repeatedly transmits the data, and the slave sends the ACK bit after every received data byte.
   - In case of the read operation, the master repeatedly receives the slave's data and sends the ACK bit after each received byte except the last one.

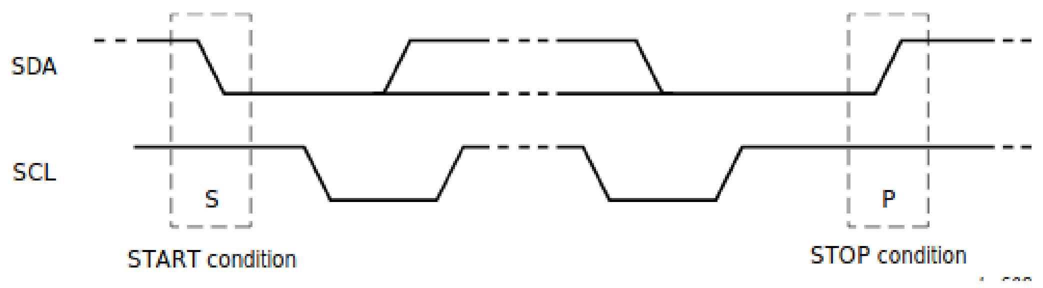
- This transaction can have multiple messages. In the case of the combined format of I2C protocol where the master has to perform multiple reads or write to one or more slaves, instead of sending a STOP bit, the master sends another START bit to retain the bus's control for another message. It is called repeated START bits.

4. If the transaction has to be ended, the master terminates it by sending a STOP bit (SDA line is set high after the SCL is set high).

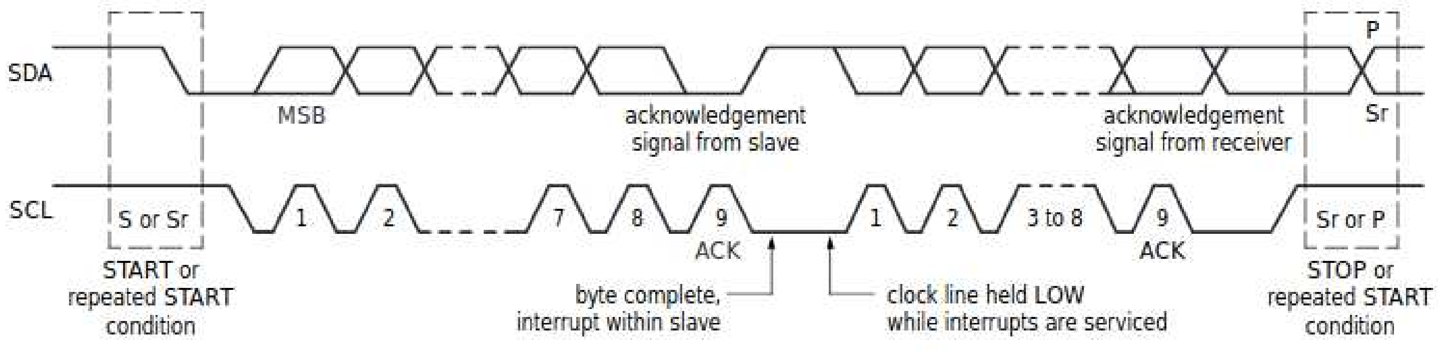
The image below shows bit transfer on the I2C bus (Source - Standard Doc).



The below image shows the START and STOP conditions on the I2C bus (Source - Standard Doc).



The below image shows the data transfer on the I2C bus (Source - Standard Doc).


SDA
SCL
MSB
acknowledgement signal from slave
acknowledgement signal from receiver
P
Sr
S or Sr
1
2
7
8
9
ACK
1
2
3 to 8
9
ACK
Sr or P
START or repeated START condition
byte complete, interrupt within slave
clock line held LOW while interrupts are serviced
STOP or repeated START condition

If the slave cannot transmit or receive another data byte due to the slave being involved in some other process like servicing some internal interrupt, it can hold the SCL line low to keep the master in a wait state. Data transfer is continued when the slave is ready to receive/transmit another byte, and it releases the SCL line.

The receiver sends the acknowledgment bit on every received byte. **ACK bit** (i.e., receiver pulls the SDA line low in the 9th clock cycle after the transfer of 8-bit data) acknowledges the transmitter that the data has been successfully received and the following data can be transferred. In the given five conditions when:

1. there is no receiver present on the bus with the transmitted address
2. receiver is busy performing some real-time functions
3. receiver receives such data or commands that it does not understand
4. receiver cannot receive any more data bytes
5. the master-receiver has to send the end of the transfer signal to the receiver

In these cases, the transmitter receives no acknowledgment signal **NACK** (i.e., the SDA line remains high during the 9th clock pulse). The master can then either send the STOP signal to end the transfer or send a repeated START signal to start a new transfer.

I2C protocol includes collision detection and arbitration to prevent data corruption in the case; multiple masters simultaneously start the data transfer. **Clock synchronization and arbitration** are used to decide which master will take control of the bus and complete the transmission when multiple masters begin transmitting on the same bus simultaneously. In the case of a single master system, it's not required. The detail working about the Clock synchronization and arbitration can be read from the standard document.

Another essential feature of the I2C protocol is **Clock stretching**. There are cases when a slave device to which the master is communicating is not ready to perform the next transaction with the master. This can be when the previous operation on the slave side has not been completed yet. In this case, the slave device can perform clock stretching, i.e., the slave can hold the SCL line low after reception and acknowledgment of byte that makes the master wait until the slave releases the SCL line to high. I2C protocol has many applications. They are used to communicate controllers with sensors, actuators, memory chips, displays, real-time clocks, digital tuning, signal processing circuits, etc. If an adversary gets access to the device's hardware part, there are several ways in which the I2C bus can be exploited if not implemented properly. In further sections in this blog, we will discuss the attacking cases and methods. The attacking methods would be very similar to what we discussed in the previous part of this blog series.

Possible Attack Scenarios

- Sniffing the communication between an I2C device like sensors, controlling device, memory chip, ADC, DAC, etc. and the controller/processor. It may lead to the stealing/leak of sensitive information by the attacker. As also discussed in the previous blog, for example, in some of the embedded devices, an I2C EEPROM is used to store some sensitive information, keys, logs, etc. This sensitive info can be leaked if an attacker sniffs the communication between the EEPROM and the controller/processor. After getting this sensitive info, an attacker can damage many ways and may be on a large scale if the vendor of the hardware has used the same sensitive info like the same keys in all the hardware in the production.
- Patching the data in the I2C based memory chips, sensors, or any controlled device. In this case, if the data is not securely managed on the device, an attacker can manipulate it. It may result in a malfunction of the device. Depending on the application, manipulation of sensor data can cause significant damage to the system.

- There are also clock glitching related attacks like modifying the frequency of the I2C clock signal. Refer to **this research paper** to get a more in-depth insight into the attacks related to I2C.

Attacking hardware via I2C

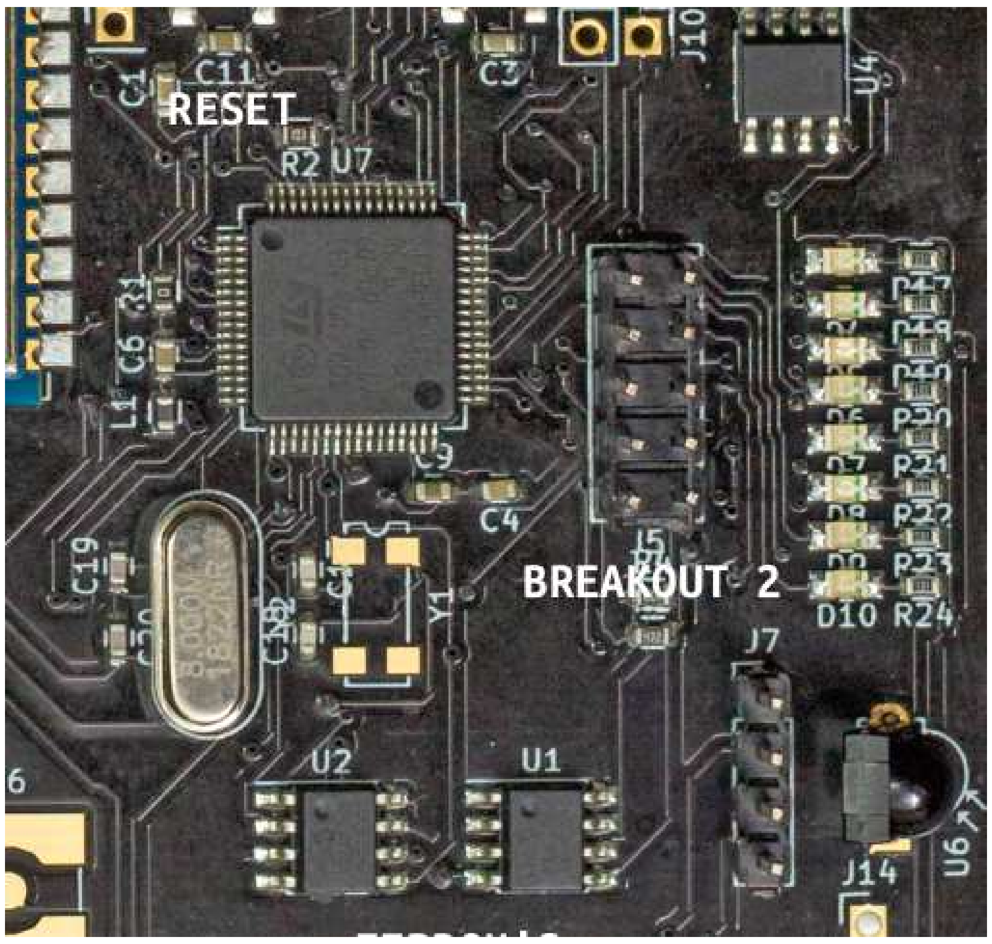
There are various ways in which the attacker can attack the hardware via I2C. Few methods to perform the attacks are explained in the below section. It is similar to that explained in the previous replacement of the blog series.

Recon

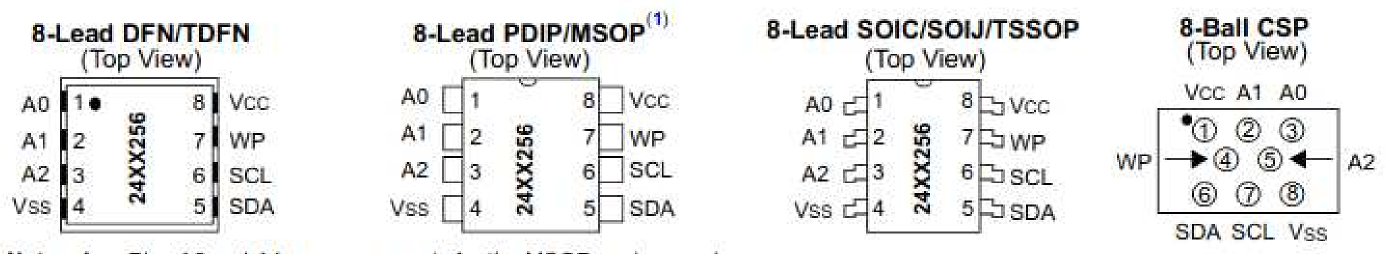
Case 1: You do not have the actual hardware, but you know the FCCID number of the device. Go to the FCCID website, search for the FCCID number of the device. If correct, you will find all the internal images and detailed internal, external information about the device. Seeing the internal image, you may get the hint for the presence of an I2C chip or any I2C based device on the hardware. Yayy!! Once you know that you have the attack surface, you can get/purchase that device and perform further required steps to attack the device.

Case 2: You got access to the hardware. Once the device's hardware is found, the first step should be to tear down the device and perform reconnaissance. Inspect each chip, test points present on the printed circuit board (PCB) to look if you can get the access to the i2c bus. You can read more about hardware recon here.

- If successful in identifying the I2C 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 I2C pins of the controller and the test points present on the PCB using **Multimeter**. For example, We are showing below the image of **EXPLIoT DIVA board** that we had also used in our previous blogs.

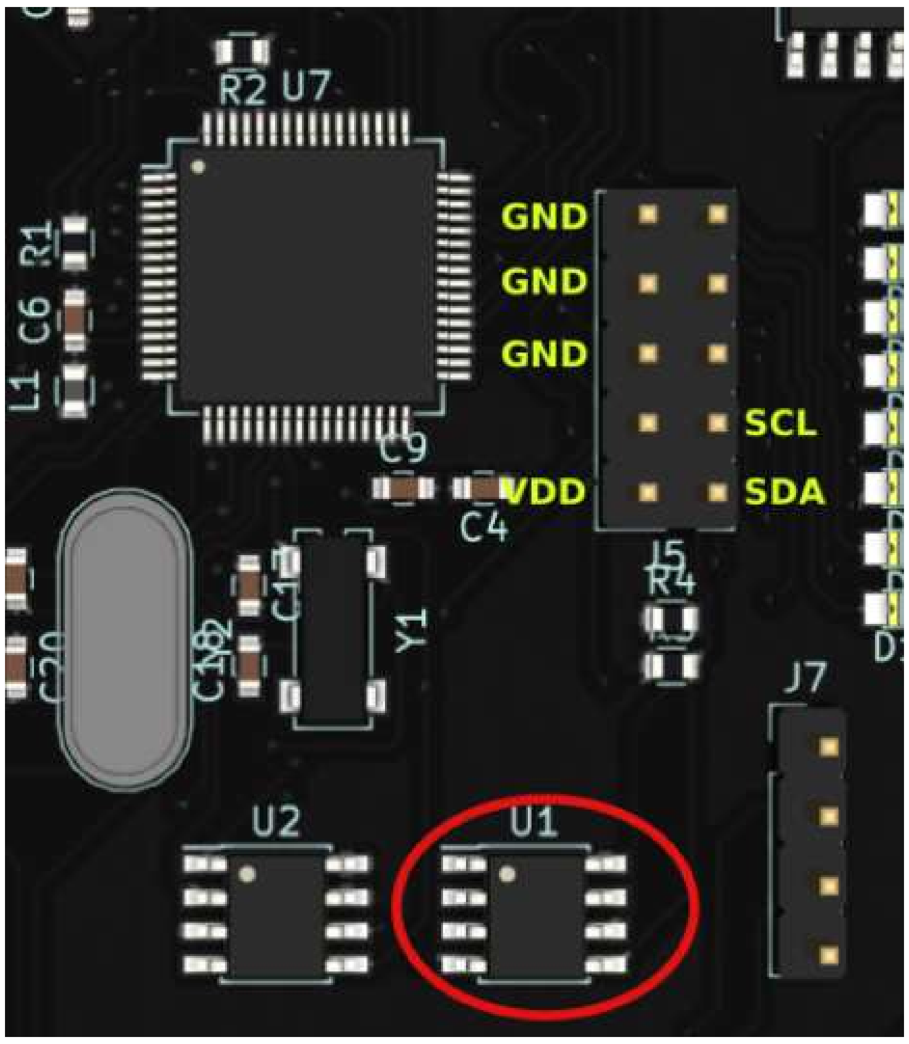


It contains some chips on the PCB. We googled the datasheet for each chip and found that one of its chips denoted as U1 is an **I2C EEPROM 24LC256** After finding the chip, the next task is to identify if the I2C pins (SDA, SCL) of the chip is connected to some points on the PCB or not. Fetching the datasheet of 24LC256, we identified the respective pins of the chip as shown in the image below (Source - 25LC256 datasheet)



Note 1: Pins A0 and A1 are no connects for the MSOP package only.

Using the multimeter, we checked the connectivity of these I2C pins with other pins on the PCB. We found that pins on the J5 header, as in the image below, were connected to these I2C pins.

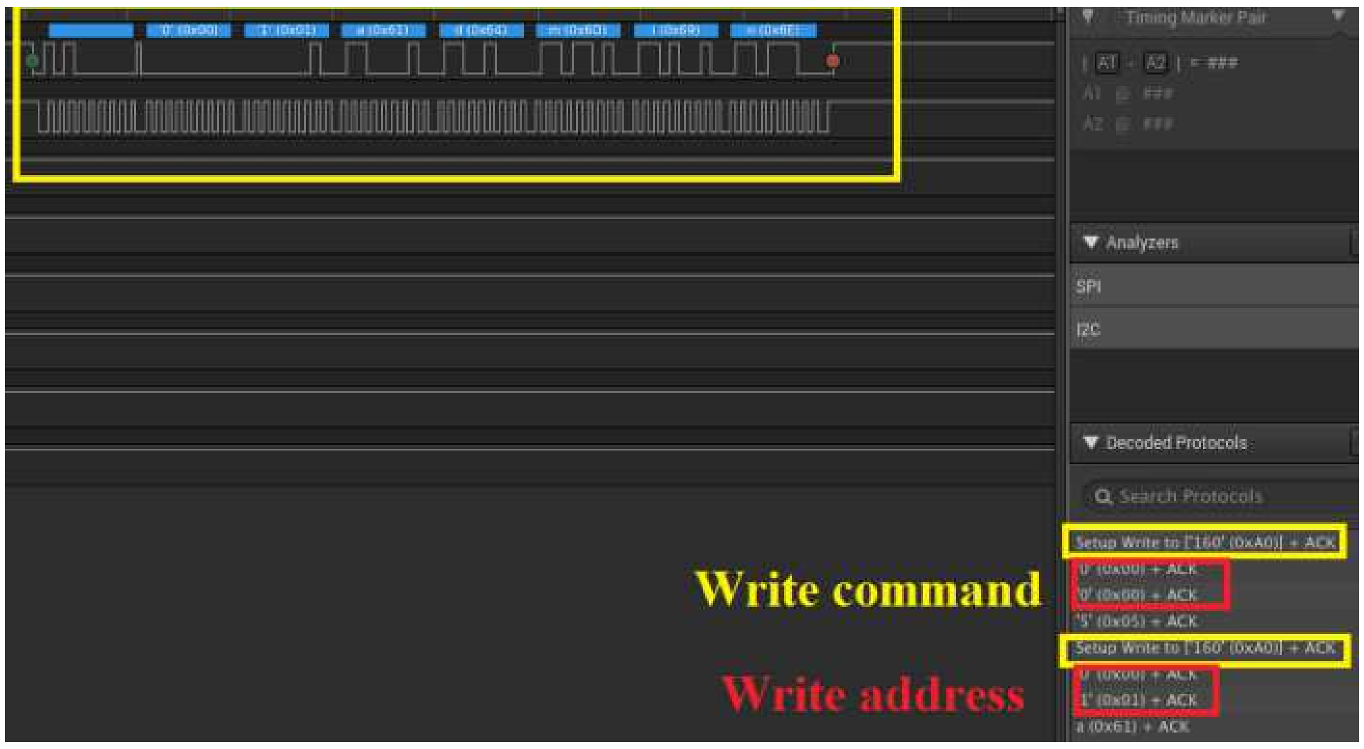


We have a goodie for you :) , EXPLIoT has Bus Auditor. It helps scanning and identifying debugging and communication interfaces exposed on any hardware board. It's demo example in case of I2C can be read in the blog here

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

Sniffing I2C Communication

Now, once the I2C pins on the hardware are successfully identified, we can go ahead and sniff the communication between the I2C chip and the controller. Tools like Logic Analyzer can be used to sniff the communication on the I2C bus.



```
Setup Write to ['160' (0xA0)] + ACK
'0' (0x00) + ACK
'0' (0x00) + ACK
'5' (0x05) + ACK
Setup Write to ['160' (0xA0)] + ACK
'0' (0x00) + ACK
'1' (0x01) + ACK
a (0x61) + ACK
```

The logic analyzer shows the signals on the SDA and SCL lines of I2C. Decode 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.

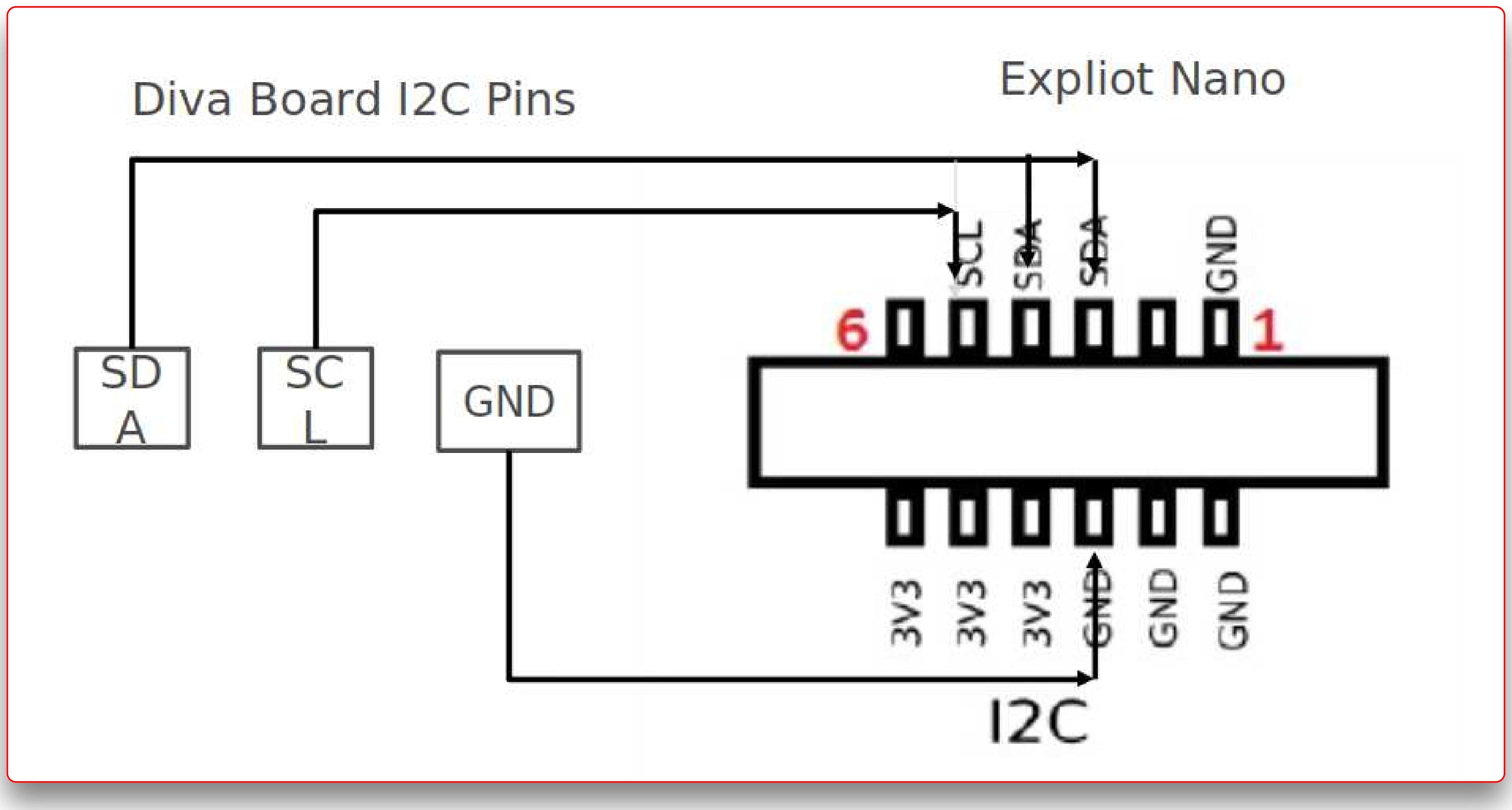
Extracting data from I2C memory Chip.

To communicate with the I2C chip present on the hardware, we need protocol adapter tools that support I2C protocol, and a framework that can make it feasible to communicate those chips without the host machine. (Sometimes, we also need to desolder the memory chips from the hardware, solder it on perf board or put it in a suitable TSOP socket and connect the required pin to the adapter.) Similar to as discussed in the [previous] blog, various tools and frameworks are available to extract the data from the memory chips. A few of them are mentioned below.

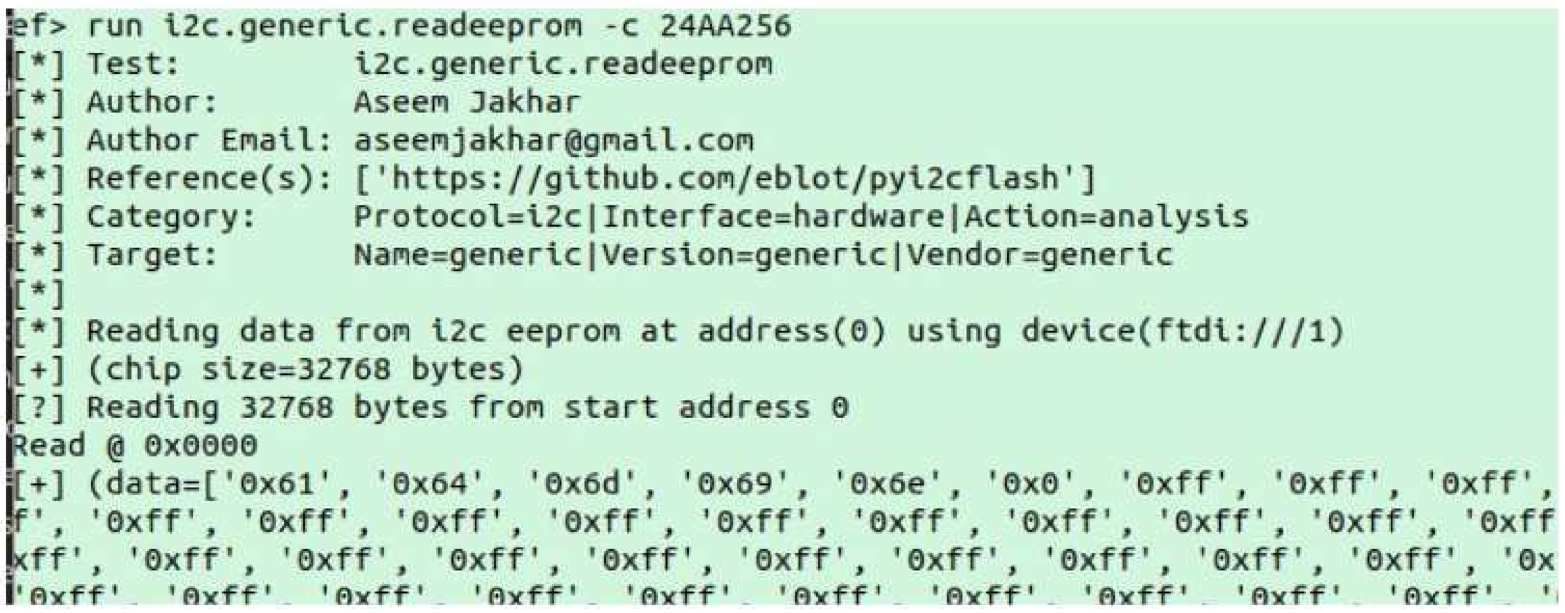
1. **EXPLIoT Nano**
2. **Bus Pirate**
3. **Shikra**
4. **CH341A**
5. **EXPLIoT Framework**
6. **pyi2cflash**
7. RaspberryPi (refer **here**) or Beaglebone (refer **here**) can also be used.

After selecting the tool and the framework that suits your requirement, check the tool's voltage levels and the chip. In case the voltage level of the adapter tool and the I2C chip do not match, use level shifters.

Case 1: You are successful in finding the tools and framework that can support your I2C memory chip After finding the I2C chip on the device, our approach would be as mentioned below. 1. Take out the datasheet of the I2C chip. Identify the pins and voltage required. 1. Accordingly, choose the adapter tool that can be used. Connect the pins of the chip to the suitable protocol 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 i2c.generic.readeeprom -c <chipname> -w <filename>` to extract the firmware from the flash memory. An example image is shown below.



```
ef> run i2c.generic.readeeprom -c 24AA256
[*] Test:         i2c.generic.readeeprom
[*] Author:       Aseem Jakhar
[*] Author Email: aseemjakhar@gmail.com
[*] Reference(s): ['https://github.com/eblot/pyi2cflash']
[*] Category:     Protocol=i2c|Interface=hardware|Action=analysis
[*] Target:       Name=generic|Version=generic|Vendor=generic
[*]
[*] Reading data from i2c eeprom at address(0) using device(ftdi:///1)
[+] (chip size=32768 bytes)
[?] Reading 32768 bytes from start address 0
Read @ 0x0000
[+] (data=['0x61', '0x64', '0x6d', '0x69', '0x6e', '0x0', '0xff', '0xff', '0xff',
f', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff
xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0x
'0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '0xff', '
```

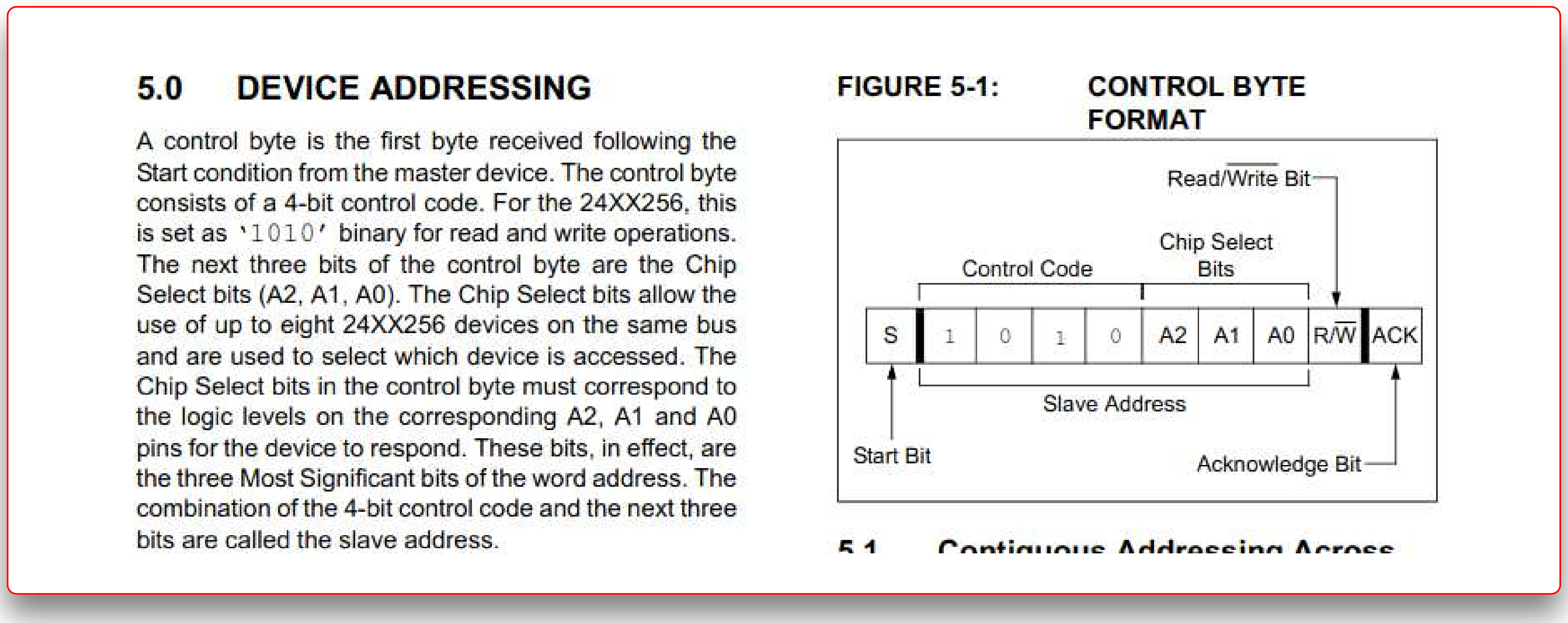
EXPLIoT Framework also supports patching of these I2C memory chips. It has plugin to write on these chips, `run i2c.generic.writeeeprom -c <chipname> -r <patched_filename>`

Similarly, other tools and frameworks as suited can be used to dump/patch the data from the I2C memory chip.

Case 2: You cannot find any tool/framework that can support your I2C memory chip In this case, the hardware setup steps would be the same as in the above case. Take any FTDI232H based adapter, match the voltage levels of the I2C 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 chips to any of the open-source frameworks of your choice. So, for this, you first need to read the datasheet of the I2C 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, for I2C memory

24LC256, let us have a glance at its datasheet. It gives the detailed instructions required to read, write, and perform other actions. The image below from 24LC256 datasheet, shows the device addressing method used in this chip.



5.0 DEVICE ADDRESSING

A control byte is the first byte received following the Start condition from the master device. The control byte consists of a 4-bit control code. For the 24XX256, this is set as '1010' binary for read and write operations. The next three bits of the control byte are the Chip Select bits (A2, A1, A0). The Chip Select bits allow the use of up to eight 24XX256 devices on the same bus and are used to select which device is accessed. The Chip Select bits in the control byte must correspond to the logic levels on the corresponding A2, A1 and A0 pins for the device to respond. These bits, in effect, are the three Most Significant bits of the word address. The combination of the 4-bit control code and the next three bits are called the slave address.

FIGURE 5-1: CONTROL BYTE FORMAT

Similarly, the below image shows I2C write operation in this chip.

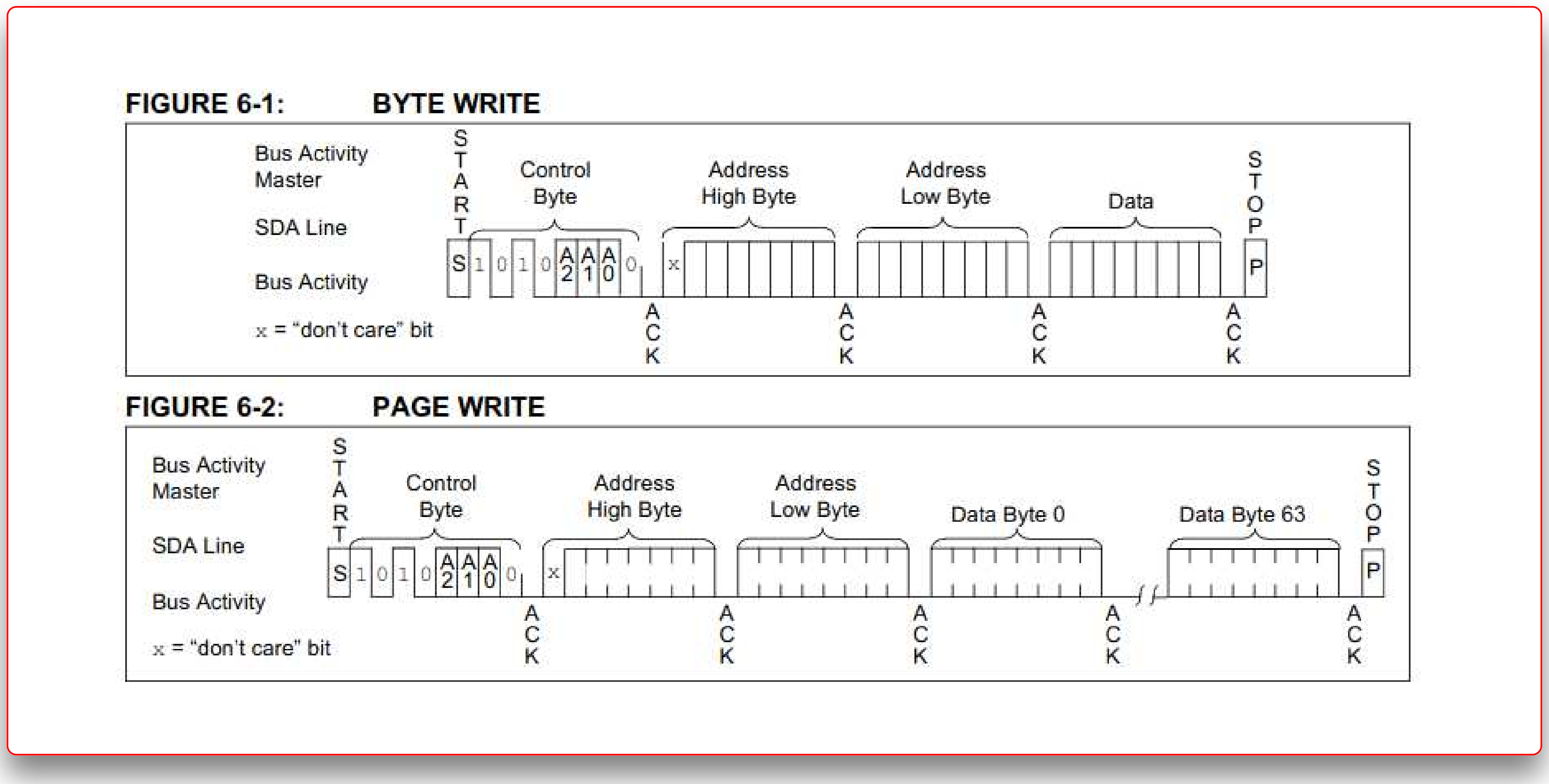


FIGURE 6-1: BYTE WRITE

FIGURE 6-2: PAGE WRITE

So, following the I2C memory chip datasheet instructions, one can write a script to dump or program the chips. In future blogs, we will come up with detailed I2C memory extraction via real-world example scenarios along with other clock glitching related attacks on the I2C bus. So, stay tuned :)

What measures can make it difficult for an attacker to attack?

- Encrypt data on the memory chip to prevent leakage of sensitive info.
- Do not hardcode sensitive data on the memory chips.
- Store encrypted data
- Physical hardening and protection of chips
- Protection against clock glitching as discussed **here** There are also a few crypto-based memory chips like **AT88SC0104CA** and some other chips as given on **Microchip website**.

Their datasheets claim to provide authentication and encryption features, and 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 I2C 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 you find an I2C device on the hardware.

Continue to the next part - IoT Security - Part 17 (101 - Hardware Attack Surface: UART)

References

- **https://en.wikipedia.org/wiki/I%C2%B2C**
- **https://www.nxp.com/docs/en/user-guide/UM10204.pdf**
- **https://www.i2c-bus.org/**
- **https://learn.sparkfun.com/tutorials/i2c**
- **https://www.researchgate.net/publication/314629854_Hardware_Attacks_on_Mobile_Robots**