Assessing and Exploiting Embedded Electronics

Version 38

801-784-2052 // justin@controlthings.io

Attacks on Field Devices

- All field devices are susceptible to embedded hardware attacks
  - Residential power/water/gas meters on are obvious targets
  - Pole-top, pipeline, and outstation devices not much harder to access (albeit riskier to your health)
- Physical defenses deter and detect, not protect
  - Locked cases and enclosures usually trivial to pick/bypass
  - Tilt and vibration alarms can be disabled before triggering
  - Cameras are often fake, or if real, not monitored
  - If alarms are triggered, response is usually measured in hours……
- If tamper/perimeter alarms are triggered, modified hardware is not easily detected

Physical Field Device Attacks

- Example of attacks to steal service from the electric utility
  - Piggybacking on other's meters
  - Meter Bypass
  - Double Feeding
  - Modify impedance levels on inputs
  - Modifying the measurement system itself
  - Changing calibration settings
- Physical attacks are not new to ICS companies
  - Represents a financial incentive to each consumer
  - ICS companies have been fighting this since day one
  - Stealing power, copper ground spike, railway steel, etc...
- However these are not cyber-attacks
  - Cyber attacks are harder to detect
  - Cyber attacks can have a much larger impact

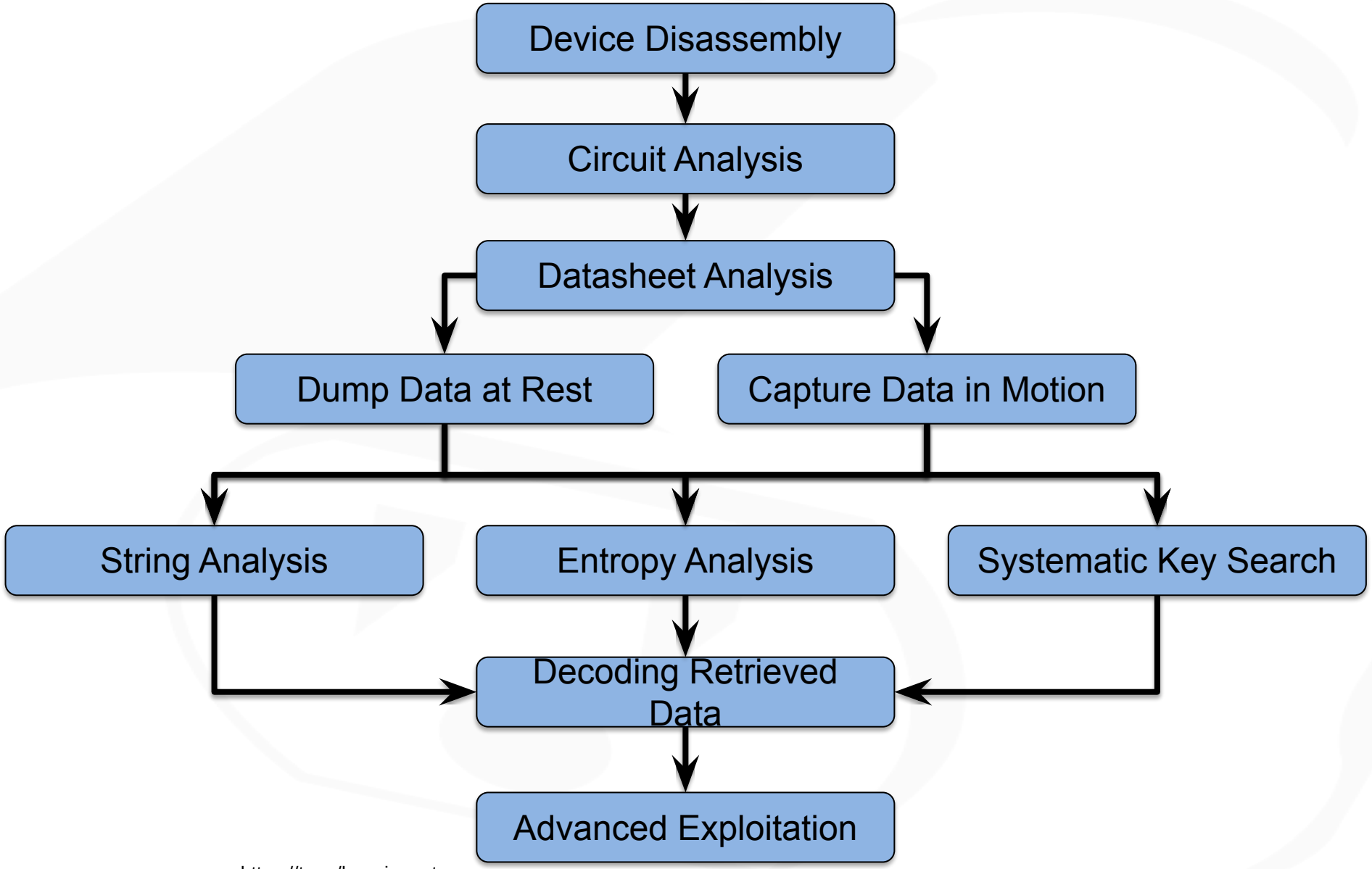
Cyber Attacks on Field Devices

- Generally only perform embedded electronic attacks when
  - Trying to bypass security controls on the device or its communications
  - Escalation of physical access to remote access
- Retrieval of config/calibration could facilitate:
  - Loss of control, reliability, or financial benefits
- Retrieval of cryptography keys could facilitate:
  - Decryption of captured network data
  - Direct access to the network
  - Impersonation of hardware device or their central control server
- Retrieval of firmware could facilitate:
  - Identification of remotely exploitable vulnerabilities
  - FHSS algorithms or cryptography key derivation routines
  - Ability to repurpose hardware as an attack tool
- Physical compromise should assume compromise of device stored data and functions

Two Primary Points of Attack

- Attacking data at rest
  - Power down the device, expose its circuit board, and interact directly with each electronic chip
  - Extract contents of accessible EEPROM, Flash, and MCU
  - Identify configs, calibrations, crypto keys, and firmware
- Attacking data in motion
  - Boot and normally operate the device in a lab while monitoring bus activity between major chips
  - Passwords often found MCU <-> maintenance interface
  - Crypto keys often be found MCU <-> crypto accelerator
  - Firmware often found MCU <-> Flash/Comm chips
  - Decrypted network traffic often found MCU <-> Comm
  - Radio config/FHSS often found MCU <-> RF chips

Embedded Electronic Methodology



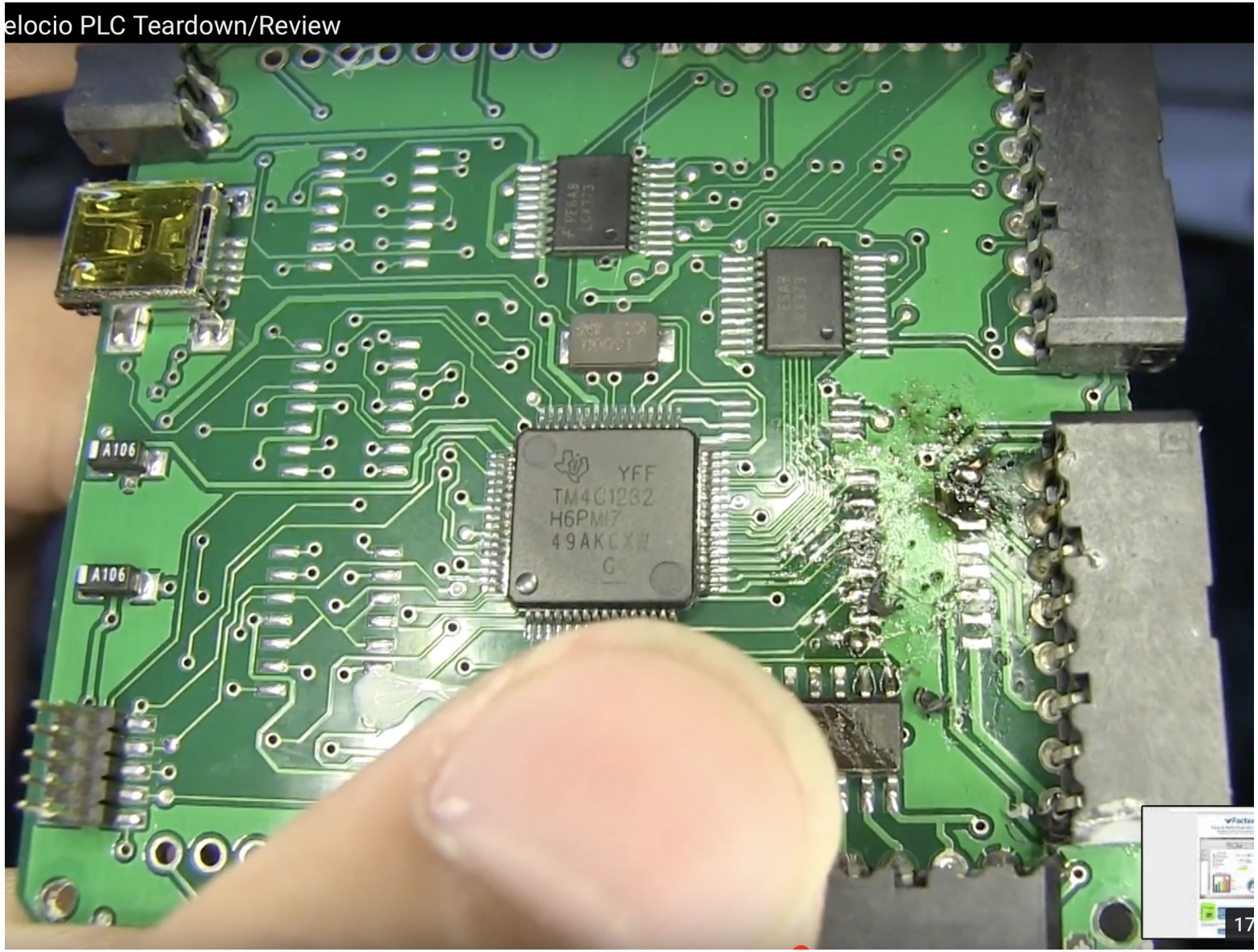
Task 1: Device Disassembly

Level of Effort: Low

Task Description: Disconnect power from the device and disassemble the device to gain access to the embedded electronic components. Attempt to do a non-destructive disassembly if possible. Document the entire process to later facilitate reassembly. Identify the existence and function of any physical tamper mechanisms protecting the device.

Task Goal: Gain physical access to embedded components and electronic buses for further testing. Identify any methods that could be used to bypass the tamper mechanisms.

Velocio PLC Circuit Board



Task 2: Circuit Analysis

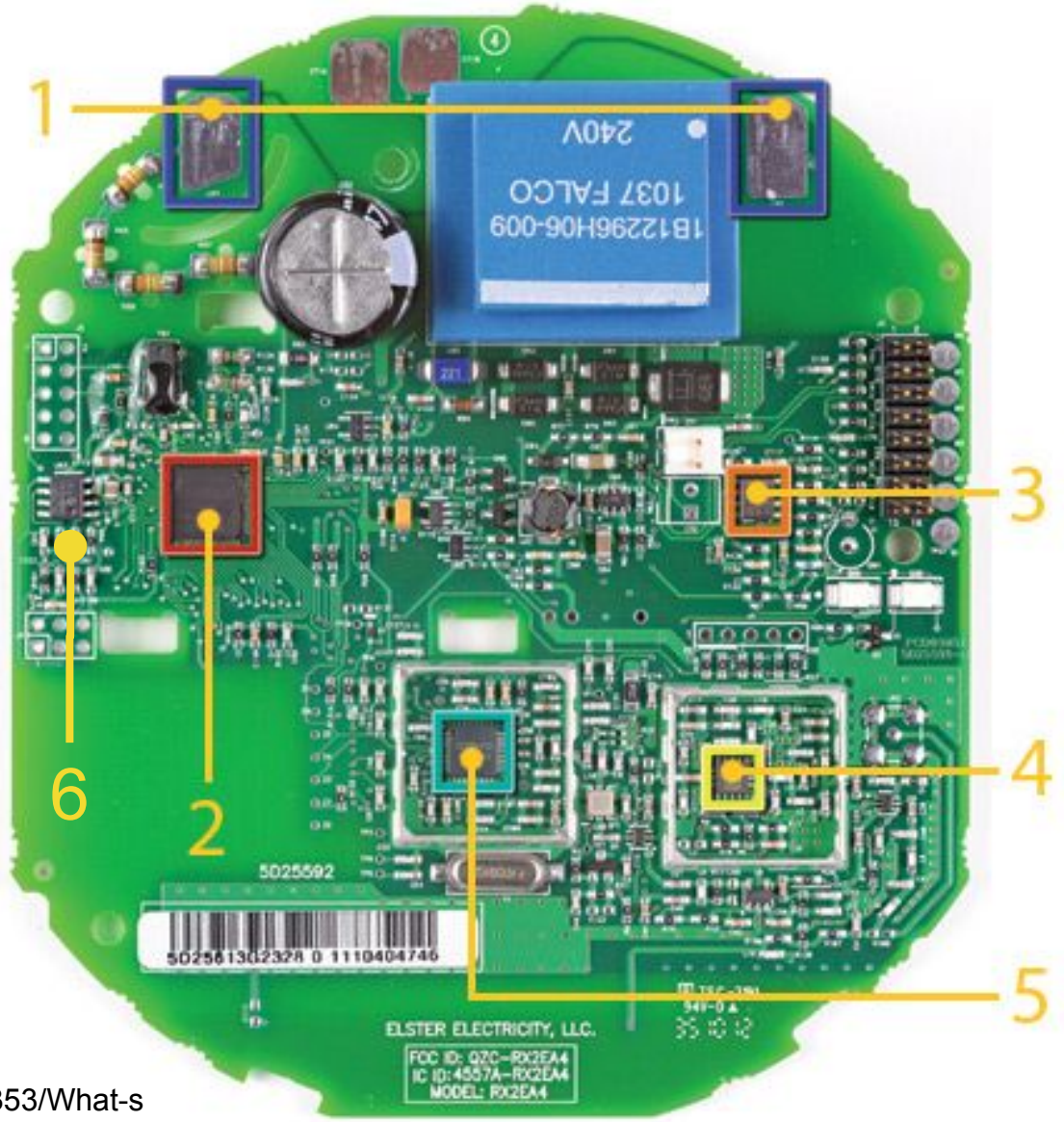
Level of Effort: Low

Task Description: Document the electronic circuit by taking pictures, reading chip IDs, tracing buses, and identifying major electronic functionality.

Task Goal: Gain information about the embedded hardware and identify potential electronic components for attack.

Smart Meter Circuit Breakdown

1. 240v Connections
2. Microcontroller (Teridian 71M6531F SOC)
3. Dual Operational Amplifier (LM2904)
4. ISM Band RF Amplifier (RFMD RF2172)
5. ISM Band RF SoC (TI CC1110F32)
6. EEPROM



Source - http://www.edn.com/design/power-management/4368353/What-s-inside-a-smart-meter-iFixit-tears-it-down

https://t.me/learninghets

Task 3: Datasheet Analysis

Level of Effort: Medium

Task Description: Find, download, and analyze all pertinent datasheets and related documentation for each major electronic component inside the device, to identify possible security weaknesses and attack angles.

Task Goal: Gain information about the function of each component and how to interface directly with each component. Identify target components and buses for following tasks.

Task 4: Dumping Data at Rest

Level of Effort: Medium

Task Description: Using the datasheets, identify the pins necessary to perform data dumping. With the device powered off, connect your testing tools and perform the dump. If needed, be sure to disable any other component by triggering reset pins or by using other methods. Review the dumped data to determine if you were successful. Attempt multiple dumps and compare the results if you are doubtful about your success.

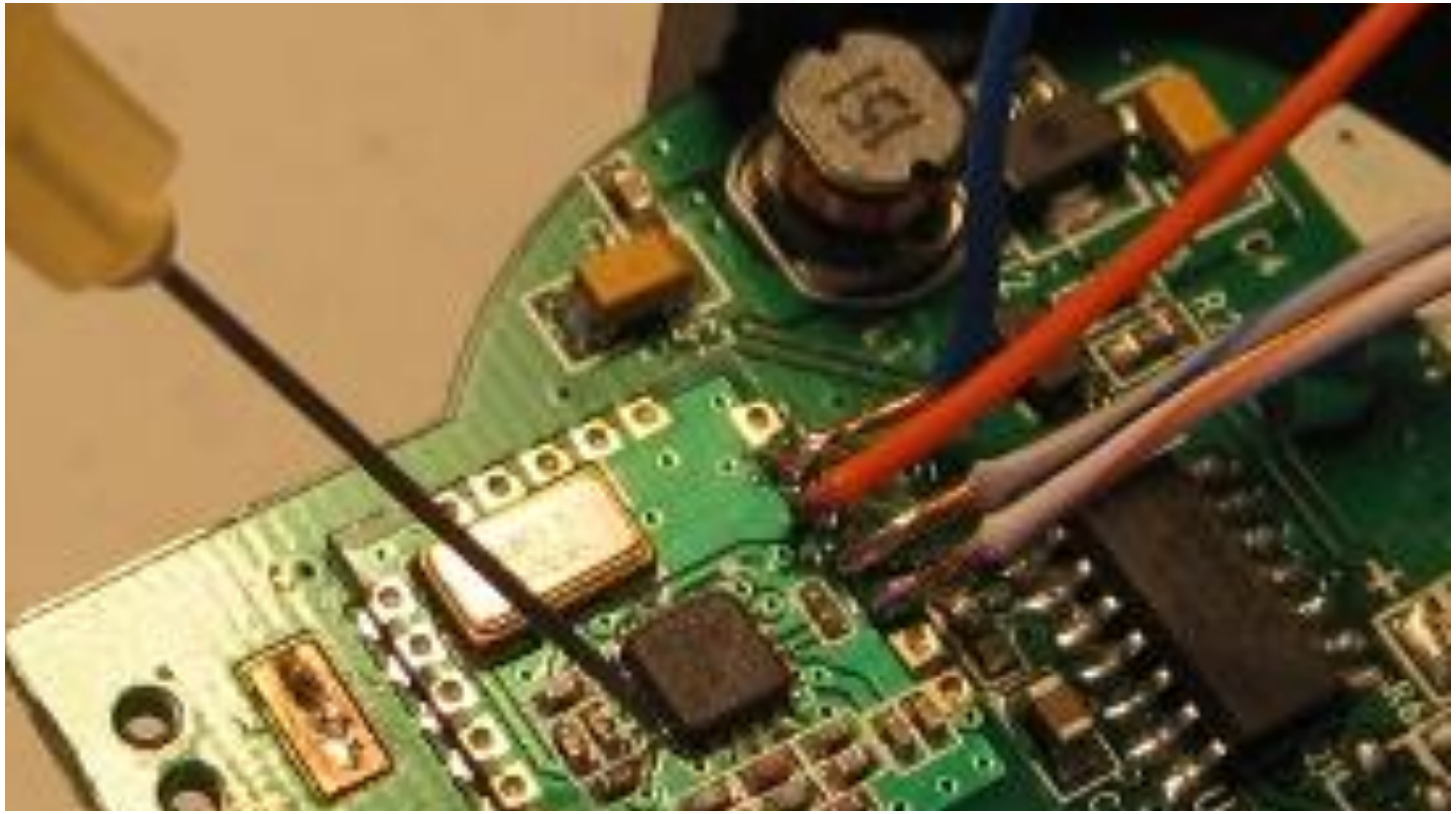
Task Goal: Obtain all data from unprotected storage devices for later analysis.

How to Dump Data at Rest

- Tools for dumping data:
  - GoodFET or GreatFET
  - Bus Pirate (See Appendix A)
  - Total Phase Aardvark
  - and many, many more...
- Steps for dumping data:
  - Power down the device you are working with and expose it circuit board
  - Wire your tool to the chip you want to dump (RAM, Flash, EEPROM, or MCU)
  - Power up your tool (not the target device) and try dumping data
  - Analyze data to make sure it is valid
  - Dump a second time to compare if you are unsure

Interfacing with an IC

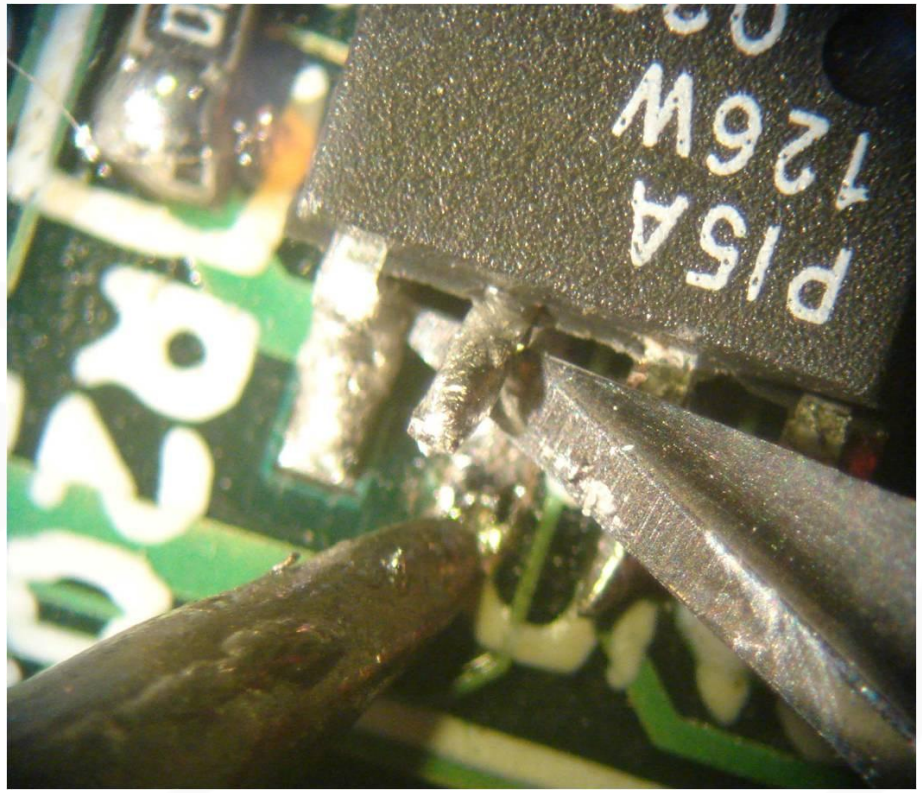
- Micro Clips: usually the best method
- Probes: great for quick dumps and captures
- Soldered wires: only when required or for reuse



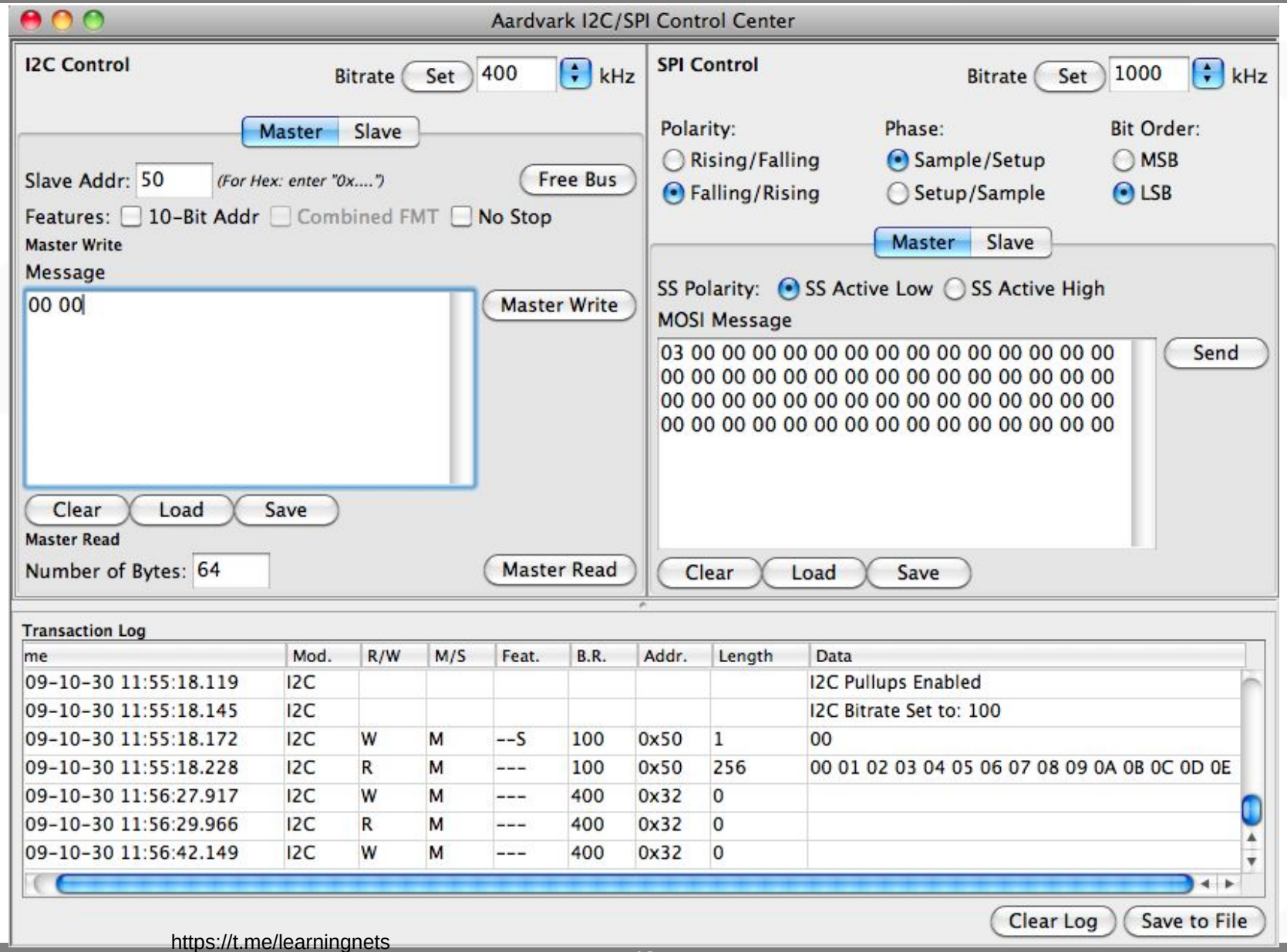
Potential Issues

- If your tool provides too much power and powers on the microcontroller
  - Tie the microcontroller's reset pin high or low

    or
  - Lift the VCC pin on target chip so power is isolated



Example: Dumping EEPROMs with TotalPhase Aardvark



Preparing our GreatFET One

- Inside your ControlThings Platform, install/update GreatFET

```
sudo pip3 install --update greatfet
sudo wget https://raw.githubusercontent.com/greatscottgadgets/greatfet/master/host/misc/54-greatfet.rules -O /etc/udev/rules.d/54-greatfet.rules
sudo udevadm control --reload-rules
```

- Plug USB0 side of your GreatFET (label on bottom) into your computer
- Virtually connect it to your VM
- Use lsusb to verify you can see your GreatFET, which appears as:

  `OpenMoko, Inc. replacement for GoodFET/FaceDancer - GreatFet`

- Run `gf info` to make sure software and hardware can speak
- Update your GreatFET flash with `gf fw --auto`
- Open a GreatFET shell with `gf shell` and try basic hardware interaction

  gf.leds[2].toggle()

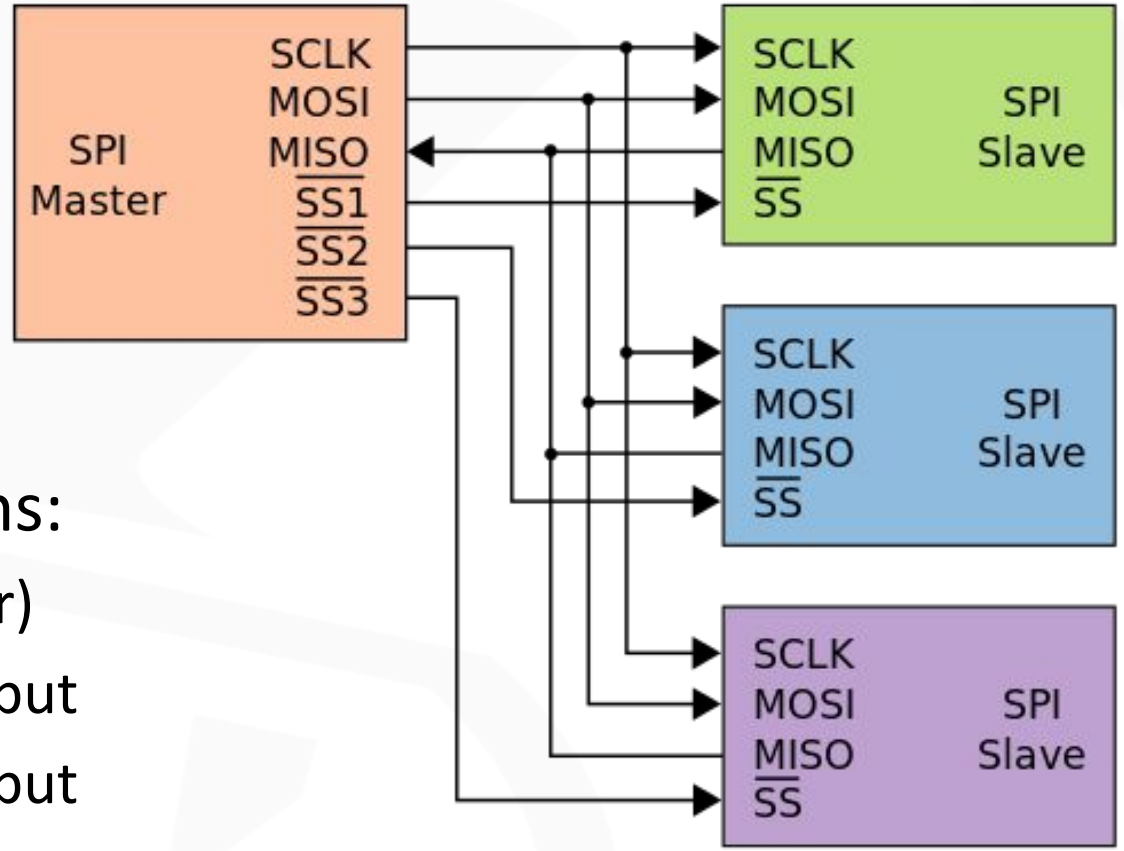
  gf.leds[2].toggle()

- You can play with tab completion after `gf.` or `exit` to leave the shell

Interacting with SPI Protocol

Serial Peripheral Interface (SPI) Bus

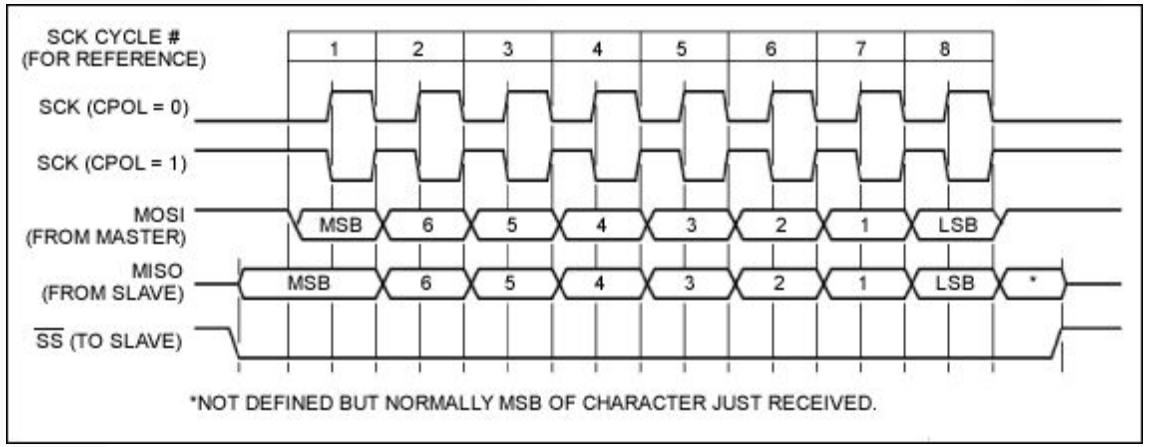
Source: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus



- The SPI bus uses four pins for communications:
  - SCLK: serial clock (from master)
  - MOSI: master output, slave input
  - MISO: master input, slave output
  - SS: slave select (from master)
- To communicate with different devices on the same bus, the master must enable the correct device by pulling that slave's SS pin low

SPI Clock Polarity and Phase Modes

- SPI has several different communications modes
  - Mode 0 (CPOL = 0, CPHA = 0)
  - Mode 1 (CPOL = 0, CPHA = 1)
  - Mode 2 (CPOL = 1, CPHA = 0)
  - Mode 3 (CPOL = 1, CPHA = 1)



Source: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

Source: http://www.maximintegrated.com/images/appnotes/4024/4024Fig02b.gif

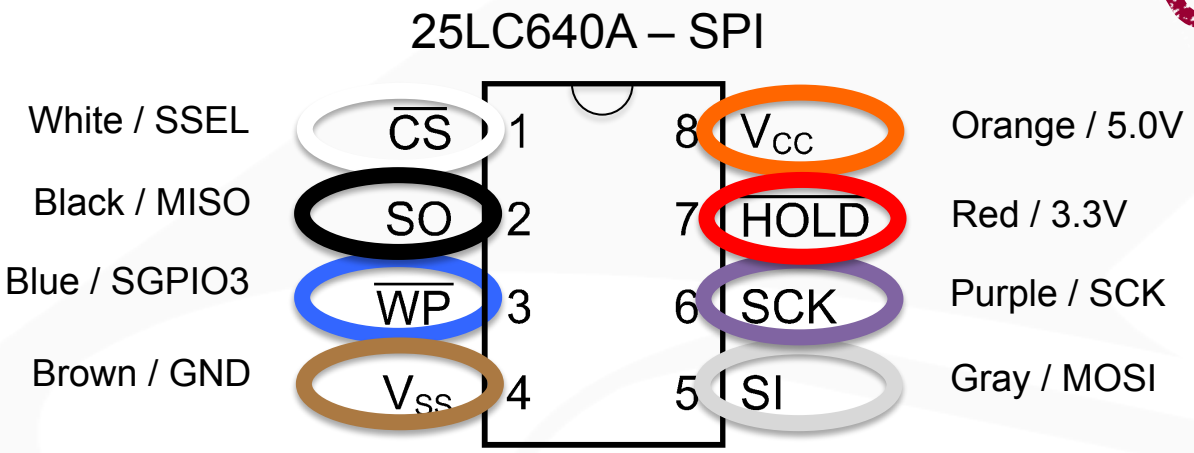
- At CPOL=0 the base value of the clock is zero
  - For CPHA=0, data is captured on the clock's rising edge (low→high transition) and data is propagated on a falling edge (high→low clock transition).
  - For CPHA=1, data is captured on the clock's falling edge and data is propagated on a rising edge.
- At CPOL=1 the base value of the clock is one (inversion of CPOL=0)
  - For CPHA=0, data is captured on clock's falling edge and data is propagated on a rising edge.
  - For CPHA=1, data is captured on clock's rising edge and data is propagated on a falling edge.

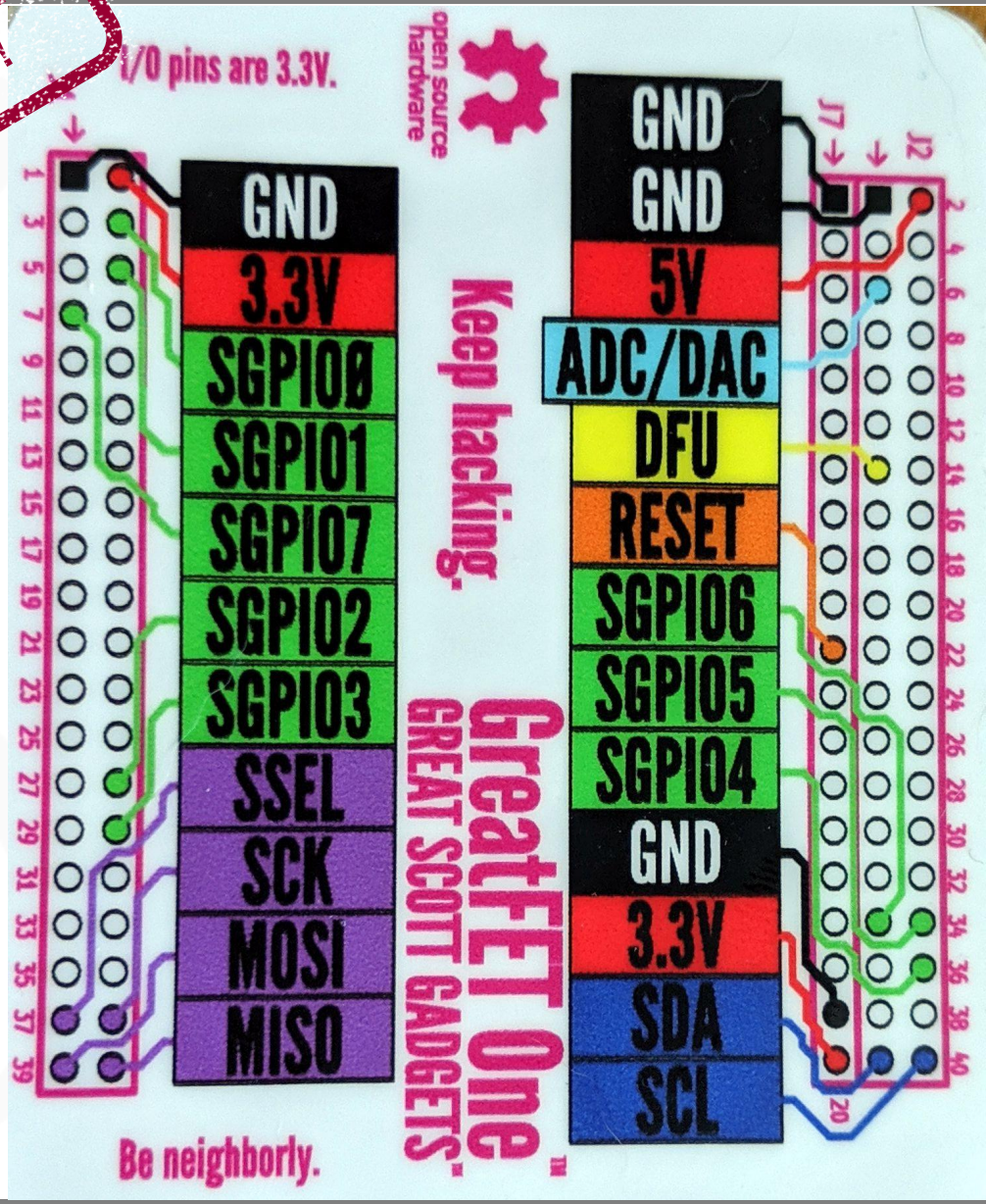
Analyzing EEPROM Datasheets

- Using the Datasheet for EEPROMS "25LC640A", find the following items:
  - 3.3v or 5v inputs
  - Protocol used to communicate
  - Maximum communication speed
  - Page size
  - Total storage size and number of blocks
  - Chip pinout diagram
  - SPI modes supported (page 5)
  - What is this chip's instruction set? (page 7)
  - How to use write protect features (page 10-11)

Wiring GreatFET to EEPROMs

- Unplug your GreatFET from USB
- Wire according to this diagram

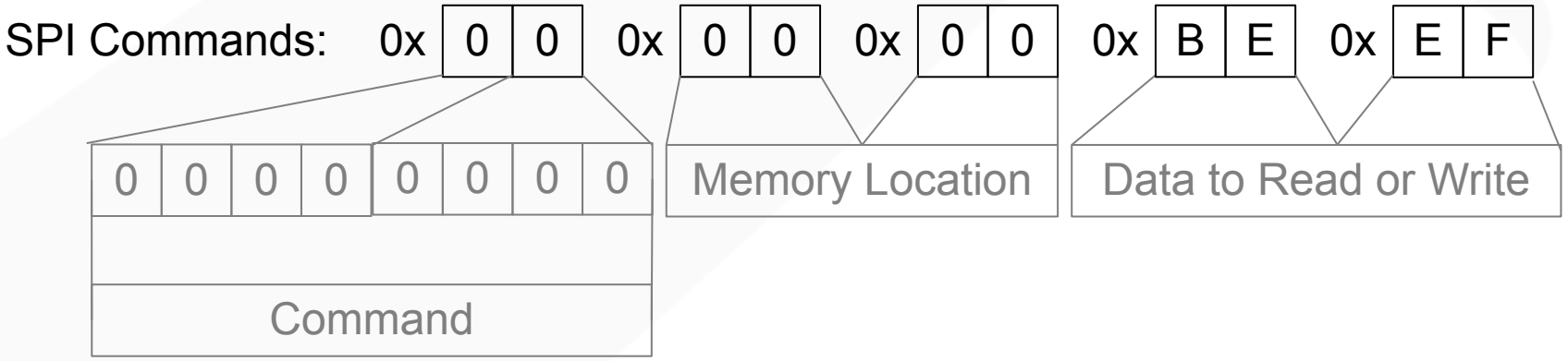




- Verify your wiring a second time
- Plug your GreatFET back into USB
- Warning!!! GreatFET does not have voltage protections on their inputs, so be careful what you connect your wires to

Our EEPROMs SPI Instruction Set

- Reading and Writing to SPI based EEPROMs:



- SPI commands (OP codes) for most EEPROMs:
  - 1: write status register (WRSR)
  - 2: write data to memory (WRITE)
  - 3: read data from memory (READ)
  - 4: disable write operations (WRDI)
  - 5: read status register (RDSR)
  - 6: write enable operations (WREN)

Speaking SPI with GreatFET

- Virtually connect your GreatFET to your ControlThings Platform VM
- Open a GreatFET terminal by running `gf shell`
- Try writing two bytes to the beginning of your EEPROM
  - `gf.spi.transmit([6])`
  - `gf.spi.transmit([2,0,0,0xBE,0xEF])`
- Now lets try read what you wrote
  - `gf.spi.transmit([3,0,0,0,0])`
- You should see:
  - `b'\xff\xff\xff\xbe\xef'` (Blue bytes are your data)
- Try writing ControlThings at address 8
  - `gf.spi.transmit([6])`
  - `gf.spi.transmit([2,0,8]+list(b"ControlThings"))`
- Now lets try read what you wrote
  - `gf.spi.transmit([3,0,0], receive_length=3+32)`

Basic Read and Write Functions

- Type the following to create a read function

```
def write(addr,data):
    gf.spi.transmit([6])
    gf.spi.transmit([2,0,addr]+list(data))
```

- Type the following to create a read function

```
def read(addr, length):
    response = gf.spi.transmit([3, 0, addr], receive_length= 3 + length)
    print(response[3:])
```

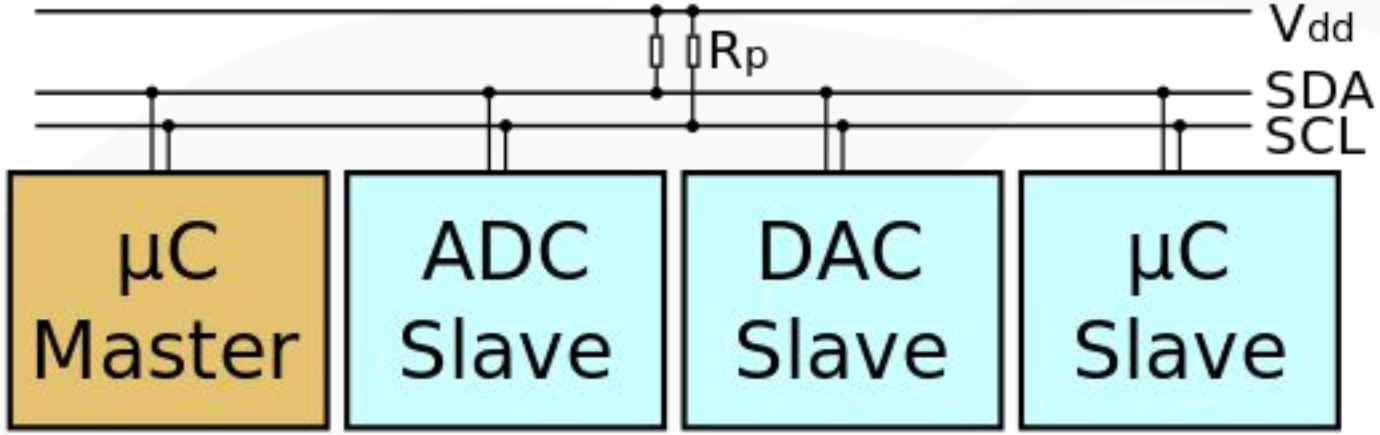
- Now try using them

```
write(32, b”This is a test!”)
read(32, 16)
```

Interacting with I2C Protocol

This Whole Section is considered Homework unless time permits

Inter-Integrated Circuit (I^2C) Bus



Source: http://en.wikipedia.org/wiki/I2C

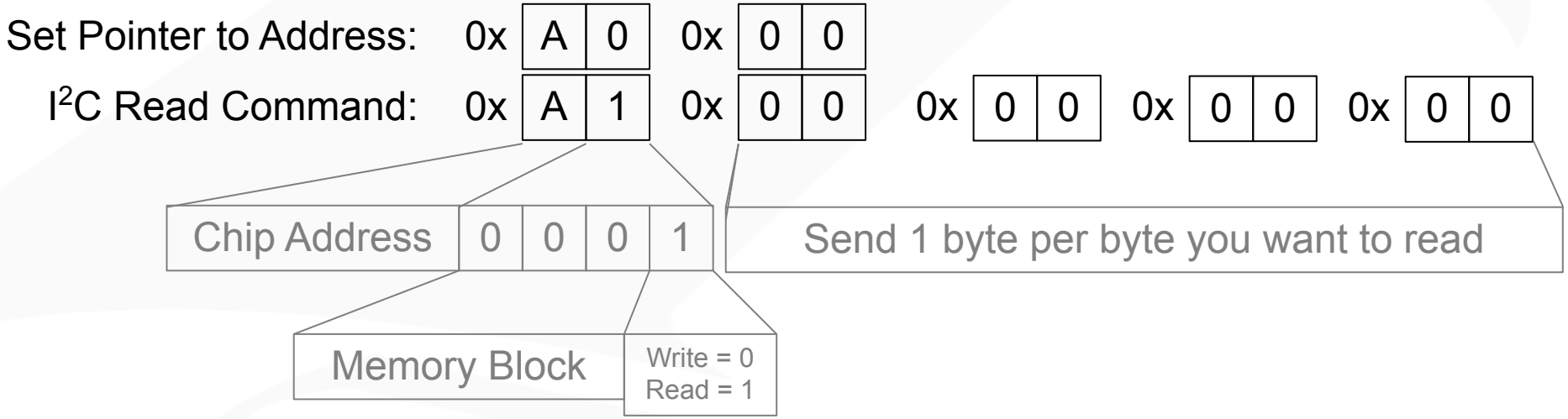
- The I^2C bus uses two pins for communications:
  - SCL: serial clock (output from master)
  - SDA: serial data line, bidirectional data
- To communicate with different devices on the same bus, the I^2C master specifies the correct slave address in its request

Analyzing EEPROM Datasheets

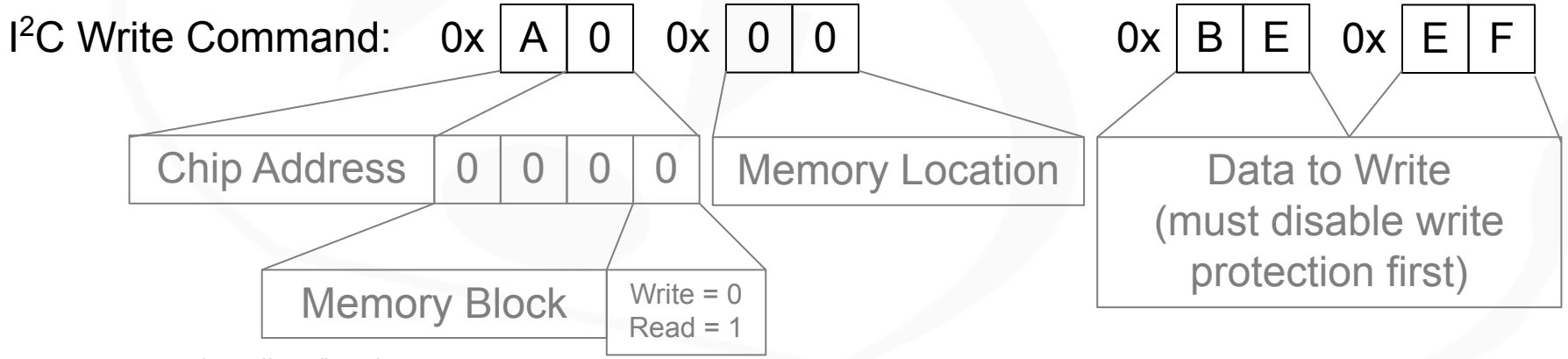
- Using your phone's camera app zoomed in, look at your EEPROMs
- Identify which of the two EEPROMs is labeled 24LC08B
- Using the Datasheet for EEPROM 24LC08B, find the following items:
  - 3.3v or 5v inputs
  - Protocol used to communicate
  - Maximum communication speed
  - Write buffer size
  - Total storage size and number of blocks
  - Chip pinout diagram
  - Chip's address or "control byte" (page 6)
  - How to use write protect features (page 8)

I²C Protocol

- Reading from I²C based EEPROMs:



- Writing to I²C based EEPROMs:



Task 5: Capturing Data in Motion

Level of Effort: Medium

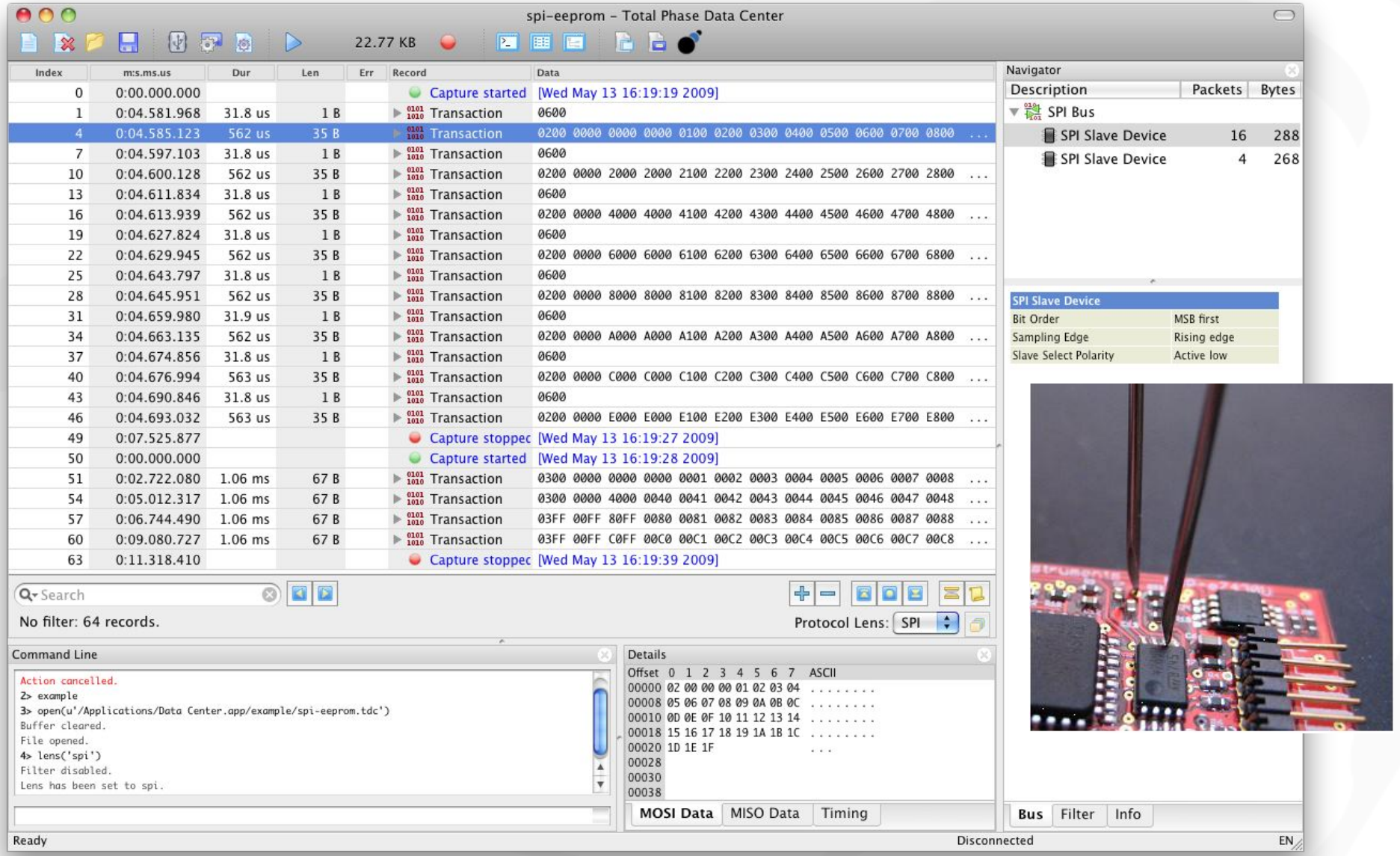
Task Description: Using the datasheets previously obtained, identify the pins and traces needed to capture bus data, sometimes called "bus snooping". With the device powered off, connect the testing tools and begin capture. Power on the device and capture sufficient data samples from each target bus. Review dumped data to identify if you were successful. Attempt multiple dumps and compare results if you are doubtful about your success.

Task Goal: Obtain data samples from all major buses for later analysis.

How to Capture Bus Data

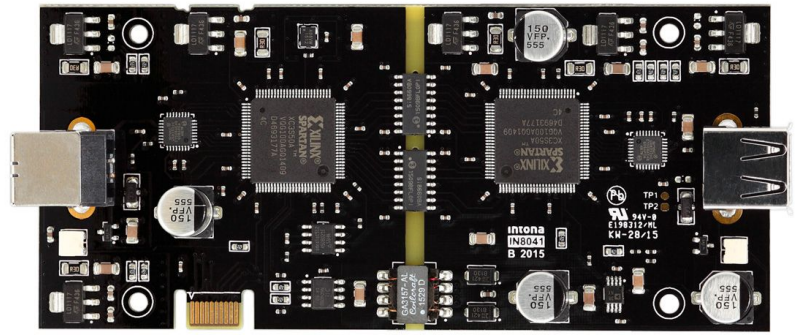
- Tools for bus capture:
  - GreatFET
  - Bus Pirate (See Appendix A)
  - Saleae Logic Analyzer (See Appendix B)
  - Total Phase Beagle
- Steps for bus capture:
  - Wire your bus sniffing tool to the busses you want to sniff
    - Crypto keys can often be found in key load operations between a microcontroller and crypto accelerator
    - Firmware can often be found in boot processes (between Flash and MCU) and firmware updates (between comms chip and MCU)
  - Start sniffer
  - Boot device and normally operate the device in a lab
- You **must** sniff **at least 4x** as fast as your target bus!

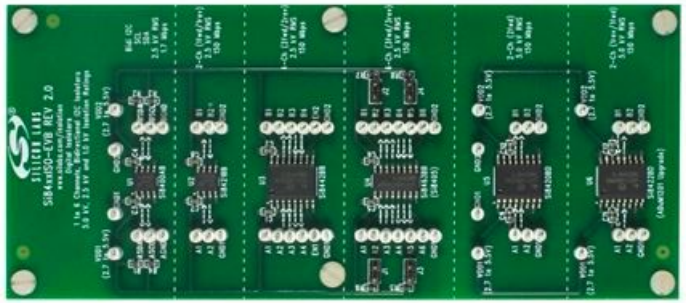
Example: Capturing Bus Data with TotalPhase Beagle



USB and Input Isolation

- Protect your laptop!
  - Connecting your USB port through a device to a live circuit can fry your USB port or even your laptop
  - Don't use the cheap Low/Full speed USB isolators, not
- Best two current options:
  - Intona UBS 2.0 High Speed (480 MBit/s) Isolator ($258)
    - http://intona.eu/en/products
    - Con: Does not protect your USB analyzer
    - Con: Has a 300mA limit so no Saleae Pro models ☹
  - SI Labs Digital Isolator Evaluation Kit ($38)
    - Digikey Link - https://goo.gl/bA2bRy
    - Pro: Protects your USB analyzing device
    - Con: Takes more effort than simple USB isolation
    - Con: Only works with digital inputs, not analog





Task 6: String Analysis of Data

Level of Effort: Low

Task Description: Use tools and multiple decoding methods to decode each obtained data. Within the logical context of the data source, identify human readable strings and other anomalies. Other identifiers may be byte patterns signifying where firmware image files begin and end.

Task Goal: Identify cryptographic keys, firmware images, and other items of interest.

Analyzing with Strings

- Open a terminal to the ~/Memory folder and use the following commands to analyze the data dumps and find the crypto key.

```
xxd memdump.bin | less
xxd -E memdump.bin | less
ghex memdump.bin
strings -eb memdump.bin | less
strings -eB memdump.bin | less
strings -el memdump.bin | less
strings -eL memdump.bin | less
strings –a –n 5 –t x memdump.bin | less
```

- How is that key encoded?
- Decode the key with Burp Suite. How do you know its a crypto key?
- Another method is with base64 command:

```
strings –n 12 memdump.bin | base64 -d > /tmp/key
binwalk /tmp/key
```

Task 7: Entropy Analysis of Data

Level of Effort: Low to Medium

Task Description: Analyze obtained data sets for blocks of data that portray high levels of entropy. Small data blocks with high entropy often signify asymmetric cryptographic keys and usually correspond to common key length sizes. Larger data blocks with high levels of entropy often signify encrypted data. Attempt to use suspected cryptographic keys to decrypt encrypted data blocks or encrypted communications traffic.

Task Goal: Identify asymmetric cryptographic keys and encrypted data objects.

Analyzing with ENT

- Ent uses an entropy rating of 0-8
  - 0 means no randomness
  - 4 means some randomness
  - 8 means high levels of randomness
- Open a terminal to the ~/Crypto directory and use the following commands to analyze the data dumps.

```
ent english-words.txt
ent english-words.txt.aes     (encrypted)
ent english-words.txt.gz      (compressed)
ent english-words.txt.bz2     (compressed)
```

- How can you tell the difference between encrypted and compressed files with ent?
- Examine the ZigBee packet capture in the ~/Memory folder. Why does it has a low entropy value?

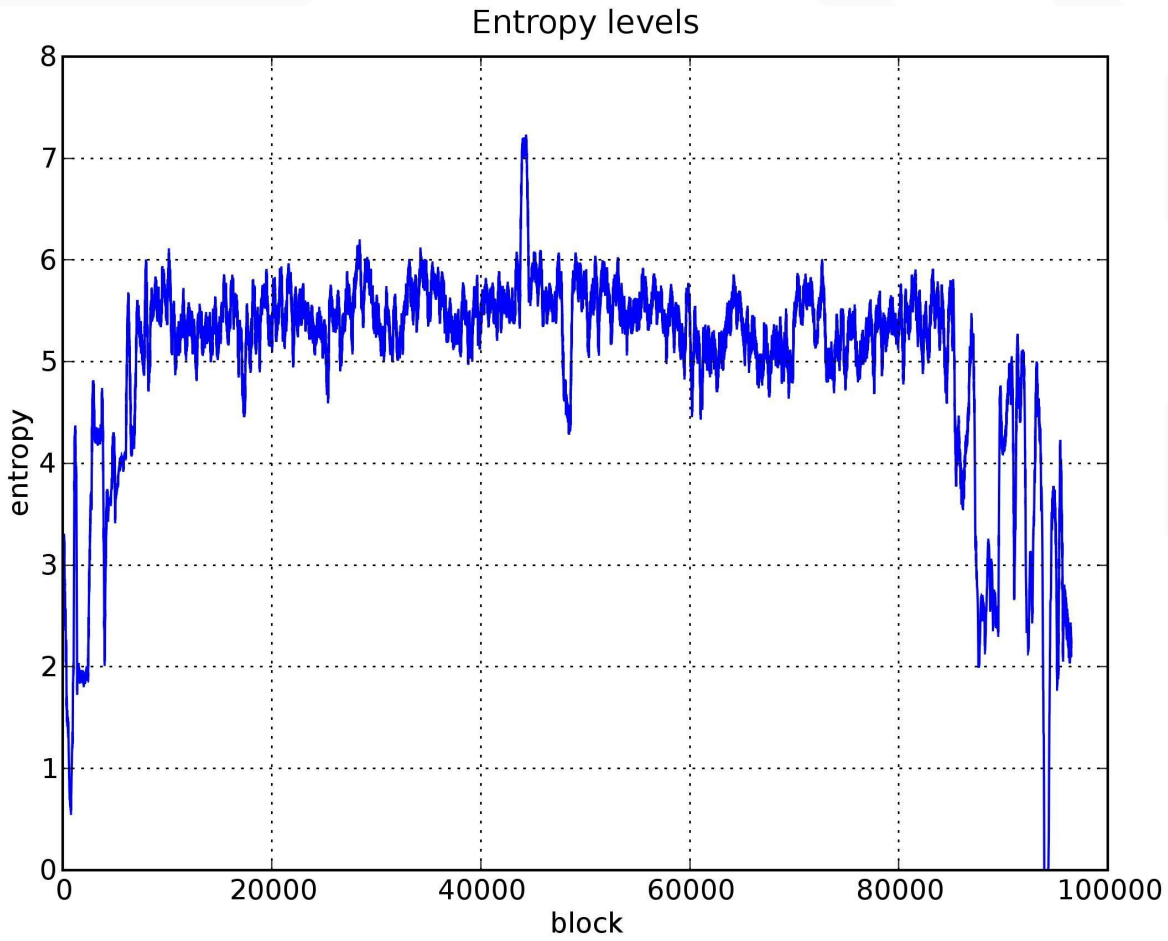
```
ent zigbee-encrypted.dcf      (two encrypted ZigBee packets in Daintree format)
ent zigbee-encrypted.pcap     (same encrypted ZigBee packets in PCAP format)
```

- Extract base64 blob from ~/Memory/memdump.bin and measure entropy

```
strings -n 12 memdump.bin | ent
strings -n 12 memdump.bin | base64 -d | ent
```

Using Entropy to Find Keys

- Asymmetric keys and randomly chosen symmetric keys have high entropy (very random)
- RAM and Flash is filled with non-random data
- Graphing entropy of flash reveals a spike in randomness
- This spike is the location of the asymmetric key in flash



Analyzing with Entropy-Graph

- Use entropy-graph to analyze the following files in the ~/Memory directory
  - hidden-key-raw
  - hidden-key-base64
- Which files can you find keys in?
- Why is it easier to find the raw key than the base64 encoded key?
- Try using "`binwalk -EB`" on both files
  - Did you see that it found the crypto key on the raw file?

Task 8: Systematic Key Search

Level of Effort: Low

Task Description: Use tools to identify cryptographic keys by attempting to use possible blocks of data from each obtained data set as the cryptographic key. For instance, if the tool is trying to identify a 128 bit symmetric key, the tool will systematically attempt to use each 128 bit data block as a potential cryptographic key to decrypt a known cryptographic artifact.

Task Goal: Identify symmetric and asymmetric cryptographic keys.

Finding Zigbee Keys in RAM Dumps

- Perform basic string searches for obvious keys
- Develop custom tools to do more advanced searches:
  - GoodFET: Abuses vulnerability in TI's Chipcon and Silicon Lab's Ember radios to access RAM even when chip is locked
  - zbgoodfind: Search for ZigBee key using RAM dump as a list of potential keys
  - Combined they can recover the ZigBee network key
- Try the following command in the ~/Memory directory


```
$ zbgoodfind -r zigbee-encrypted.pcap -f memdump.bin
zbgoodfind: searching the contents of memdump.bin for encryption keys
with the first encrypted packet in zigbee-encrypted.pcap.
Key found after 7077 guesses:  c0 c1 c2 c3 c4 c5 c6 c7 c8 c9 ca cb cc cd
ce cf
```

Task 9: Decoding of Retrieved Data

Level of Effort: High

Task Description: Reverse engineering of the data in an attempt to understand its purpose. For instance, testers could attempt to understand the captured data blocks to determine what each set of bytes represent in the serial bus protocol or the data stored in the flash/EEPROM chips. This could be done by sending known commands or setting known configurations and attempting to identify in the data blocks where those commands and configurations are transmitted and stored.

Task Goal: Identify the purpose of blocks of data that could be used in exploitation attempts.

Contact Information

Justin Searle
personal: justin@meeas.com
work: justin@inguardians.com
cell: 801-784-2052

twitter: @meeas
Facebook: www.facebook.com/m33as
LinkedIn: www.linkedin.com/in/meeas
GitHub: github.com/meeas, github.com/ControlThingsTools

Control Things Platform Releases:

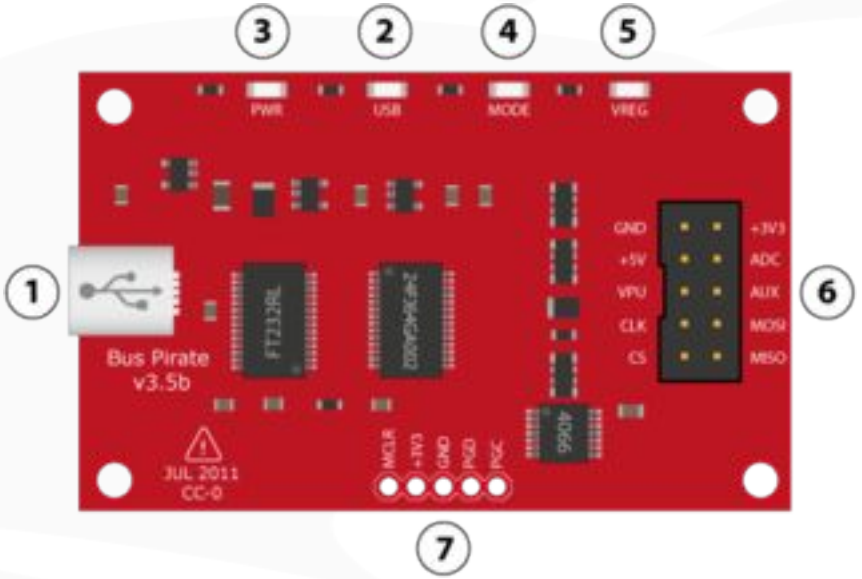
https://goo.gl/wwqxqb

Appendix A: Using Dangerous Prototype's BusPirate

This section is based on BusPirate 3.6, but should be mostly applicable for newer versions.

Warning, the 4.x line of BusPirates has been out for years, but has never had a feature complete firmware and considered by most as being Beta quality.

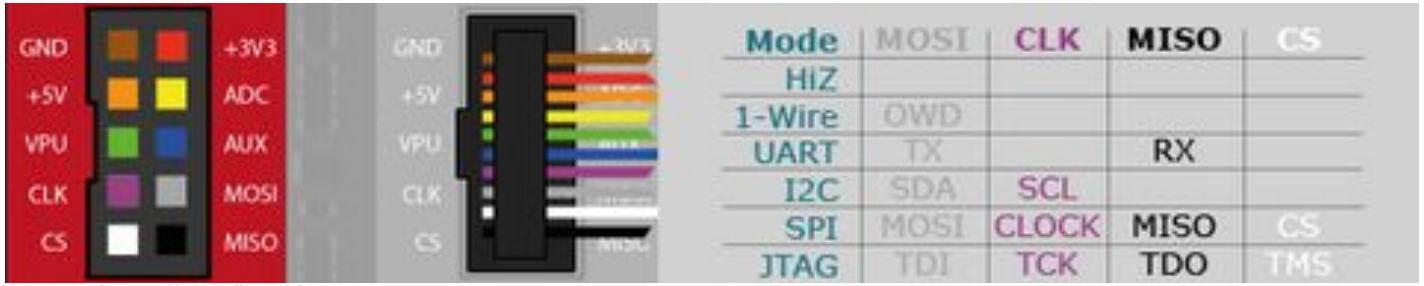
Bus Pirate 3.6



1. **Mini-B USB Port**: Connects the Bus Pirate to a PC
2. **USB Transmit Indicator**: LED flashes when there's traffic from the PIC to the PC
3. **Power Indicator**: LED lights when Bus Pirate is powered by USB
4. **Mode Indicator**: LED lights when Bus Pirate is configured for a protocol mode from the user terminal (menu 'm')
5. **Voltage Regulator Indicator**: LED lights when on-board power supplies are activated from the user terminal (command capital 'W')
6. **I/O pins**: Connects the Bus Pirate to external circuits

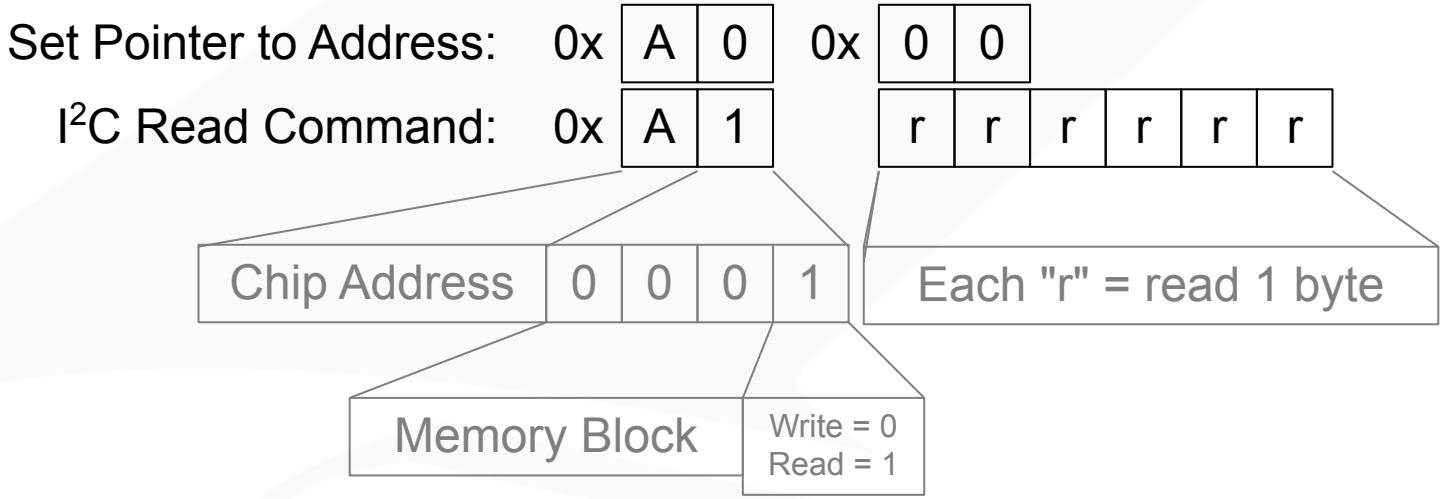
Bus Pirate Pinout

	Pin Name	Description (Bus Pirate is the master)
GRY	MOSI	Master data out, slave in (SPI, JTAG), Serial data (1-Wire, I2C, KB), TX* (UART)
PUR	CLK	Clock signal (I2C, SPI, JTAG, KB)
BLK	MISO	Master data in, slave out (SPI, JTAG) RX (UART)
WHI	CS*	Chip select (SPI), TMS (JTAG)
BLU	AUX	Auxiliary IO, frequency probe, pulse-width modulator
YEL	ADC	Voltage measurement probe (max 6volts)
GRN	Vpu	Voltage input for on-board pull-up resistors (0-5volts).
RED	+3.3v	+3.3volt switchable power supply
ORG	+5.0v	+5volt switchable power supply
BRN	GND	Ground, connect to ground of test circuit

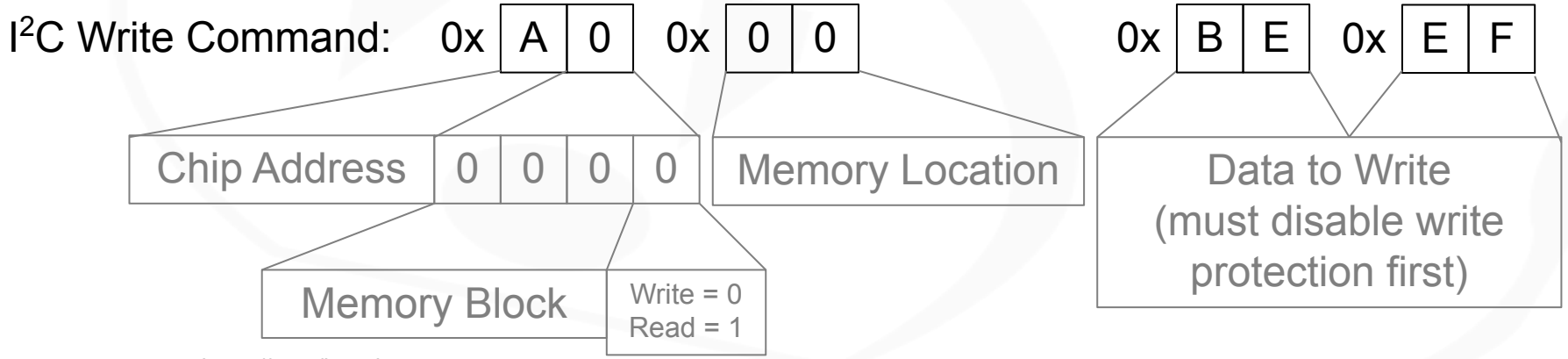


Bus Pirate's I²C Protocol

- Reading from I²C based EEPROMs:

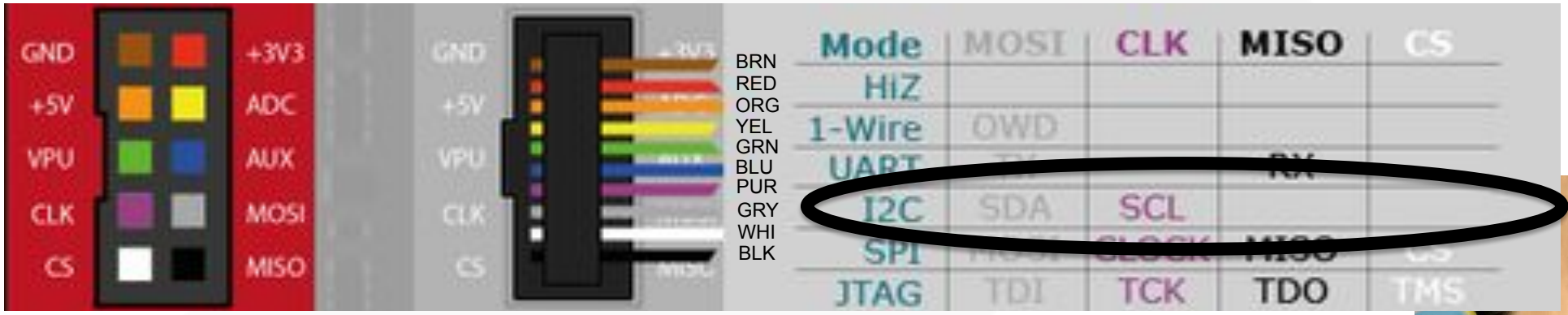


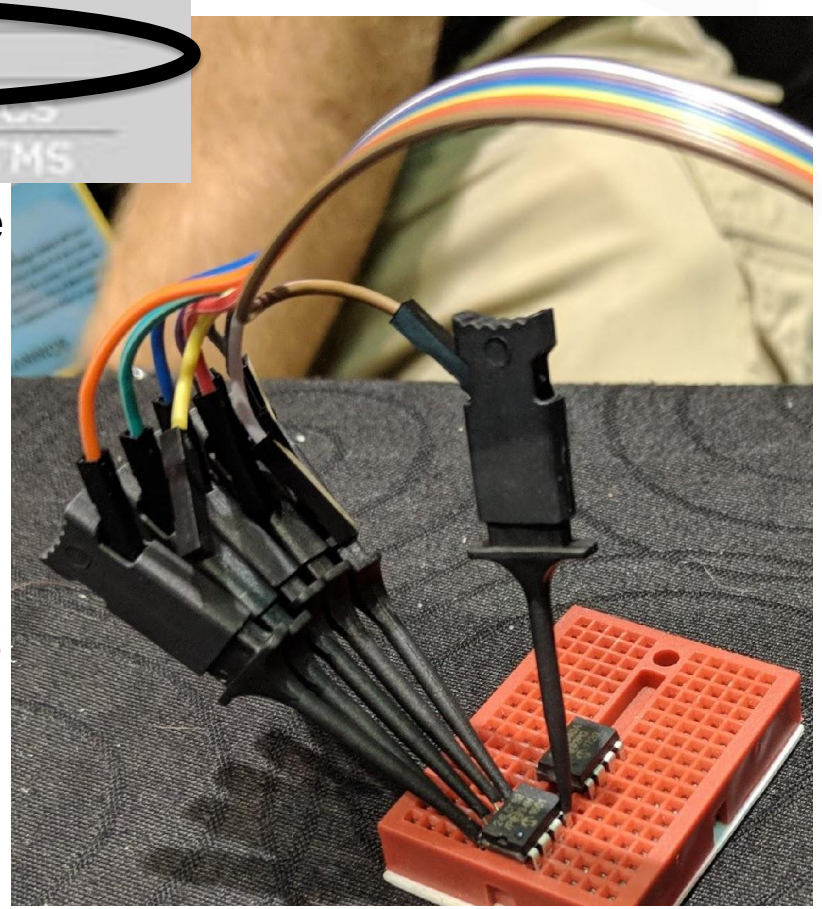
- Writing to I²C based EEPROMs:



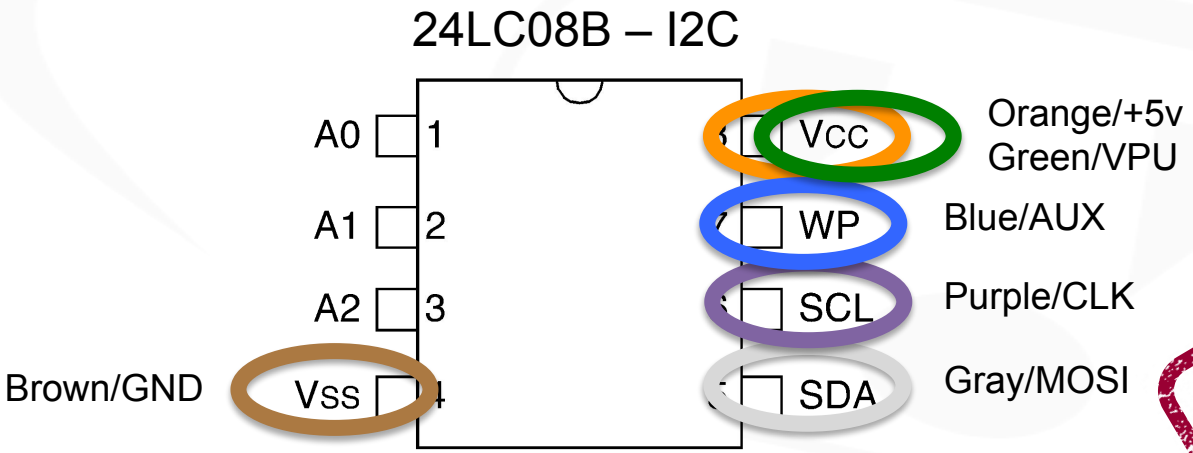
Bus Pirate to I²C Based EEPROMs

From Bus Pirate Documentation at dangerousprototypes.com/docs/Bus_Pirate





Note: All colors represent Bus Pirate probe wire colors not the color of the probe



Reading from I2C Based EEPROMs

- Plug the Bus Pirate into your computer and it connected to your VM
- From Activities, start the buspirate application and hit enter a few times
- Hit the enter key a few times until you see a prompt
- Talking on the I2Cbus (use `?` to see help screen)
  - Type `m` to display mode menu
  - Type `4` to use I2C mode
  - Type `4` to select ~400KHz
  - Type a capital `W` to enable the Bus Pirate's power supply
  - Type a capital `P` to enable pull-up resistors
  - Type a capital `A` to take AUX high to enable Write Protection
  - Type `(1)` to see if you are wired correctly to the EEPROM.
    If it doesn't show of list of device addresses, check your wiring because it is probably incorrect
  - Type `[0xA0 0]` to reset pointer to address 0
  - Type `[0xA1 r:1024]` read full 1KB EEPROM
- You should see a lot of `0xFF` since your EEPROM is empty

Writing to I2C Based EEPROMs

- Write some data to address 0 and read it back
  - Type `a` to take AUX low to disable Write Protection
  - Type `[0xA0 0 0xBE 0xEF]` to write 2 bytes at address 0
  - Type `[0xA1 r:8]` to read 8 bytes. Why don't you see the `0xBE 0xEF`? Lets try reading it the right way...
- Reset your pointer to read from the correct address
  - Type `[0xA0 0]` to read reset your pointer to address 0
  - Type `[0xA1 r:8]` to read 8 bytes
  - Now you should see the `0xBE 0xEF`, if not, verify AUX is low and check your wiring
- Try writing and read some other data now
  - Type `[0xA0 16 "Control"]` to write 7 bytes at address 16
  - Type `[0xA0 16][0xA1 r:7]` to read 7 bytes from address 16
  - Type `o4` to change output to raw output mode to show ASCII
  - Type `[0xA0 16][0xA1 r:7]` to read 7 bytes from address 16 again
  - Type `[0xA0 48 "abcdefghijklmnopqrst"]` to write 20 bytes
  - Type `[0xA0 48][0xA1 r:20]` to read 20 bytes from address 48
  - Why is the data at address 48 show `qrstefghijklmnop`? How big is your write buffer?
  - Type `o1` to change output back to hex output mode

Understanding I²C memory blocks

- Bus Pirate's I²C mode supports macros

```
I2C>(0)
 0.Macro menu
 1.7bit address search
 2.I2C sniffer
```

EEPROM Address	Block and Address	Write Command
0-255	Block 0, Address 0-255	0xA0
256-511	Block 1, Address 0-255	0xA2
512-767	Block 2, Address 0-255	0xA4
768-1023	Block 3, Address 0-255	0xA6

- Macro 1 attempts to discover available blocks

Use command 0xA0 to Write to block

Use command 0xA2 to Write to block

0
Use command 0xA1 to **R**ead to block

1
Use command 0xA3 to **R**ead to block

```
I2C>(1)
Searching I2C address space. Found devices at:
0xA0(0x50 W) 0xA1(0x50 R) 0xA2(0x51 W) 0xA3(0x51 R)
0xA4(0x52 W) 0xA5(0x52 R) 0xA6(0x53 W) 0xA7(0x53 R)
0xA8(0x54 W) 0xA9(0x54 R) 0xAA(0x55 W) 0xAB(0x55 R)
0xAC(0x56 W) 0xAD(0x56 R) 0xAE(0x57 W) 0xAF(0x57 R)
```

- Why did it find 8 blocks when the datasheet said 4?

SELF GUIDED LAB!!!

Continue with I²C Based EEPROMs

EEPROM Address	Block and Address	Write Cmd	Read Cmd
0-255	Block 0, Address 0-255	0xA0	0xA1
256-511	Block 1, Address 0-255	0xA2	0xA3
512-767	Block 2, Address 0-255	0xA4	0xA5
768-1023	Block 3, Address 0-255	0xA6	0xA7

- Reference the table to the right for the following tasks
- Write to Block 2 (block 0b010)
  - Type `a` to take AUX low to disable Write Protection
  - Type `[0xA4 0][0xA5 r:8]` to reset pointer to address 0 on the third block and read 8 bytes on that block
  - Type `[0xA4 0 "Control"]` to write 7 bytes at address 0
  - Type `[0xA4 0][0xA5 r:8]` to reset pointer to address 0 and read 8 bytes
- Read full 1KB of EEPROM starting from the Block 3, looping around
  - Type `[0xA6 0][0xA7 r:1024]` to reset pointer to address 0 on Block 3 and read 1k bytes from that point
- How would you write Things to EEPROM address 812?
  - Type `[0xA6 44 "Things"]`
  - Type `[0xA6 44][0xA7 r:8]`
- Quit the Bus Pirate terminal by unplugging the Bus Pirate

Scripting Dumps of I2C Based EEPROMs

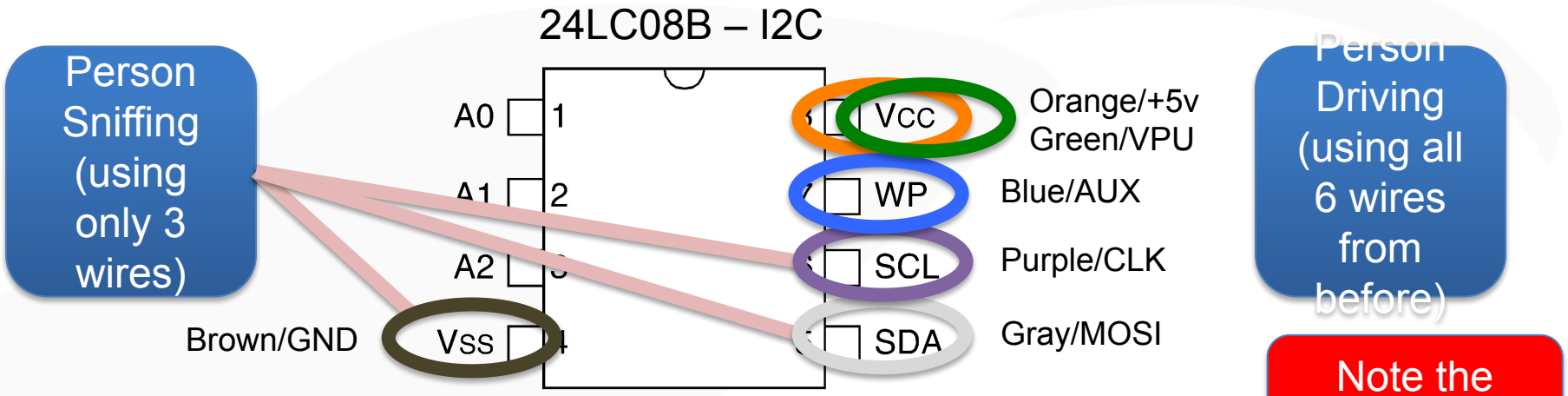
1. Unplug your bus pirate and plug it back in to make sure it resets
2. From a bash terminal, dump the full EEPROM to a binary file using i2c-dump

   ```
   i2c-dump -s 1024 -b 256 -o /tmp/24LC08B.bin
   ```

   – This will take about 30 seconds to run
3. Use ghex or xxd to read the file you just created which should be the contents of your EEPROM

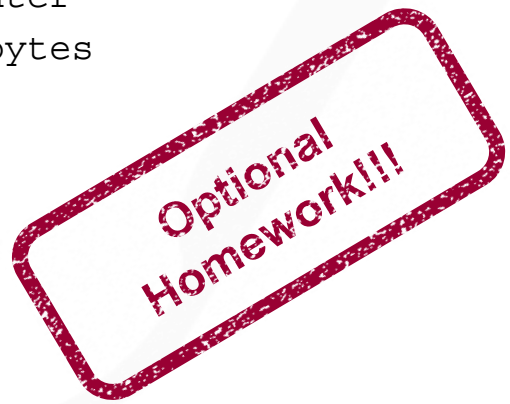
   ```
   ghex /tmp/24LC08B.bin
   xxd /tmp/24LC08B.bin | less
   ```

I²C Capture with a Second BusPirate



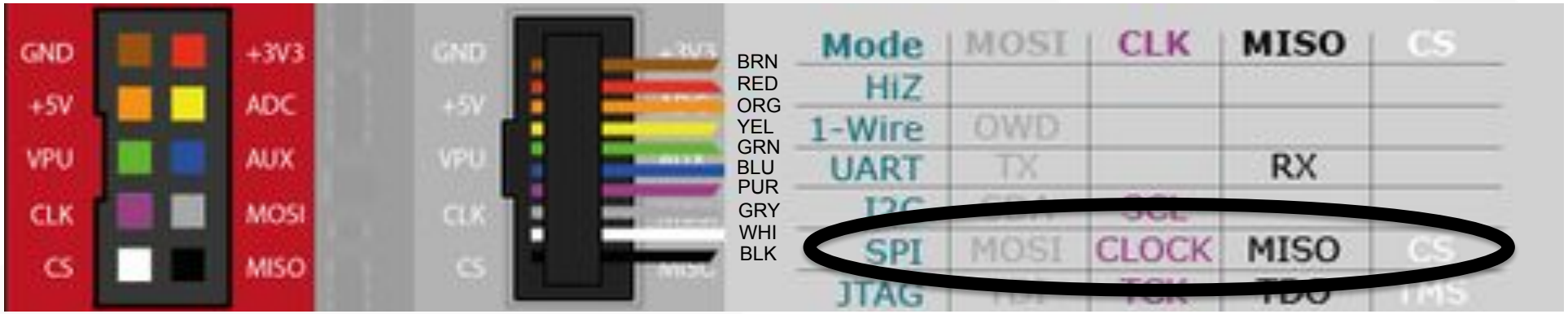
- Sniffing from the I2C bus
  - Type "m4" to use I2C mode
  - Type "4" to select ~400KHz
  - Type "(2)" to enter sniffing
  - Remember, you are using the Bus Pirate Terminal menu item, it automatically logs your session to your home folder. If you are using the screen command, don't forget to use the -L option to log

- Talking on the I2C bus
  - Type "m4" to use I2C mode
  - Type "3" to select ~100KHz
  - Type "WPa" to setup power and pins
  - Type "[0xA0 0 0xBE 0xEF]" to write
  - Type "[0xA0 0]" to reset pointer
  - Type "[0xA1 r:8]" to read 8 bytes



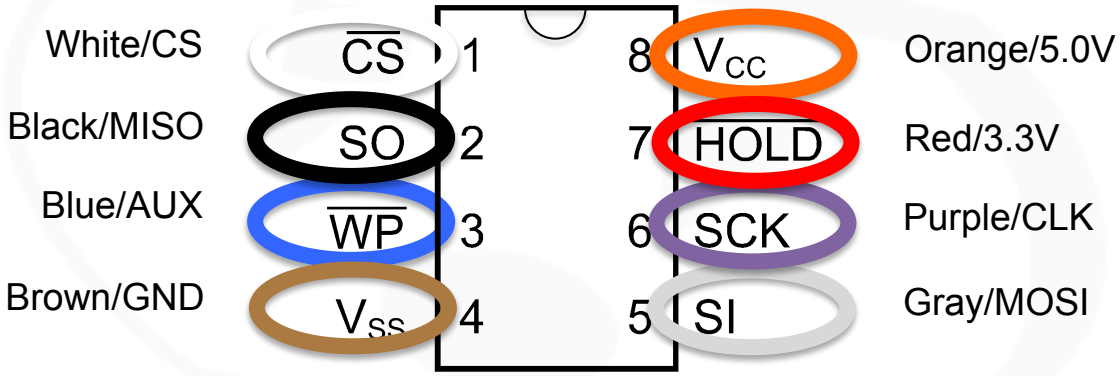
Bus Pirate to SPI based EEPROMs

From Bus Pirate Documentation at dangerousprototypes.com/docs/Bus_Pirate



Note: All colors represent Bus Pirate probe wire colors not the color of the probe

25LC640A – SPI



NOTE: EEPROMS that lack pull-up resisters require connecting Pin 7 to Pin 8 ... or cheat like we did above ...

Working with SPI Based EEPROMs

- Connect Bus Pirate to your computer and verify it connected to your VM
- From Activities, start the buspirate application and hit enter a few times
- Talking on the SPI bus (use ? to see help screen)
  - Type m to display mode menu
  - Type 5 to use SPI mode
  - Use the following options: 4,1,2,2,2,2
  - Type W to enable the Bus Pirate's power supply
  - Type A to disable Write Protection
  - Type [6][1 0b00000010] set the write enable latch (WEL) bit in register
    See page 10 in datasheet and table 2.2 on that page
  - Type [3 0 0 r:8] to read 8 bytes
  - Type [6] to enable writes via WREN command
  - Type [2 0 0 0xBE 0xEF] to write 2 bytes to address 0
  - Type [3 0 0 r:8] to read what you wrote
  - Type [6][2 0 2 "Control Things"] to write 14 bytes to address 2
  - Type [3 0 0 r:32] to read what you wrote

[6] must be issued before EVERY write!

Check wiring if you can't read 0xBE 0xEF

Writing to SPI chips

- Read the entire EEPROM
  - Type `[3 0 0 r:8192]` (8KB = 8 x 1024 = 8192)
- Now see if it loops back to address 0 if you read too much
  - Type `[3 0 0 r:8200]`
  - Can you see what you wrote earlier in address 0?
- How would you write I want to control all the ICS things with the 32 byte page limit?
  - Type [6][2 0 0 "I want to control all the ICS th"]
  - Type [6][2 0 32 "ings"]
- How would you write Control Things to address 4100?
  - Type [6][2 16 4 "Control Things"]

16 x 256 = 4096
0 x 1 = 4
4096 + 4 = 4100

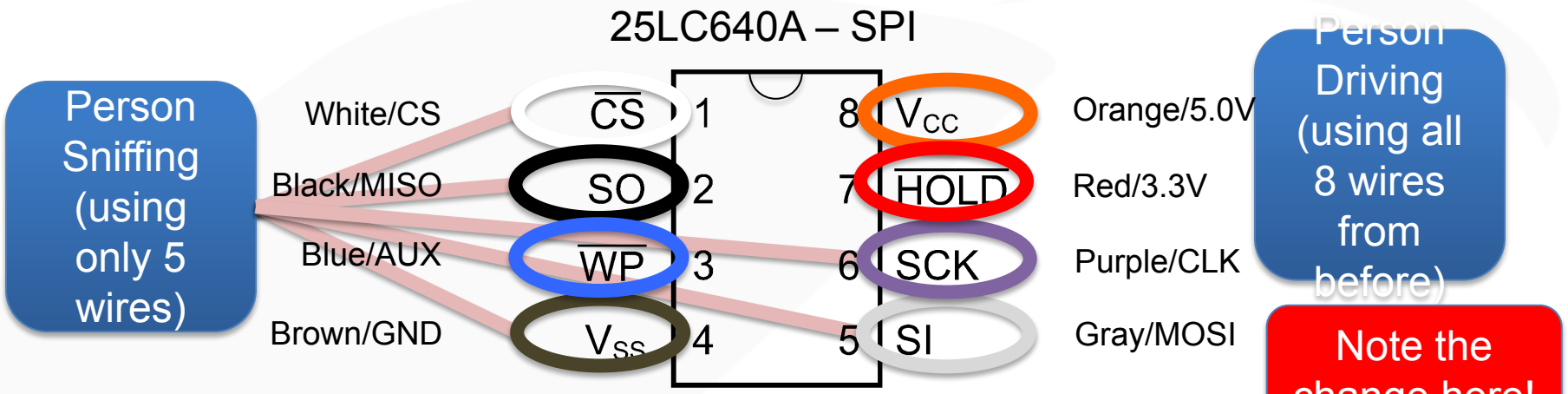
- Unplug your bus pirate and plug it back in to make sure it resets
- From the terminal (not buspirate interface), dump the full EEPROM to a binary file using spi-dump

```
spi-dump -s 8192 -o /tmp/25LC640a.bin
```

- Use `ghex` or `xxd` to read the file you just created which should be the contents of your EEPROM

```
ghex /tmp/25LC640a.bin
xxd /tmp/25LC640a.bin | less
```

SPI Capture with a Second BusPirate



- Sniffing from the SPI bus
  - Type "m5" to use SPI mode
  - Use the following options: 4,1,2,2,2,2
  - Type "(1)" to enter sniffing
  - Remember, you are using the Bus Pirate Terminal menu item, it automatically logs your session to your home folder. If you are using the screen command, don't forget the -L option to log

- Talking on the SPI bus
  - Type "m5" to display modes
  - Use the following options: 1,1,2,2,2,2
  - Type "W" to enable power supply
  - Type "A" for AUX enable writes
  - Type "[1 2]" to enable latch (WEL) bit
  - Type "[6]" to enable WREN command
  - Type "[2 0 0 0xBE 0xEF]" to write
  - Type "[3 0 0 r:8]" to read

Optional Homework!!!

Appendix B: Using Saleae Logic Analyzers

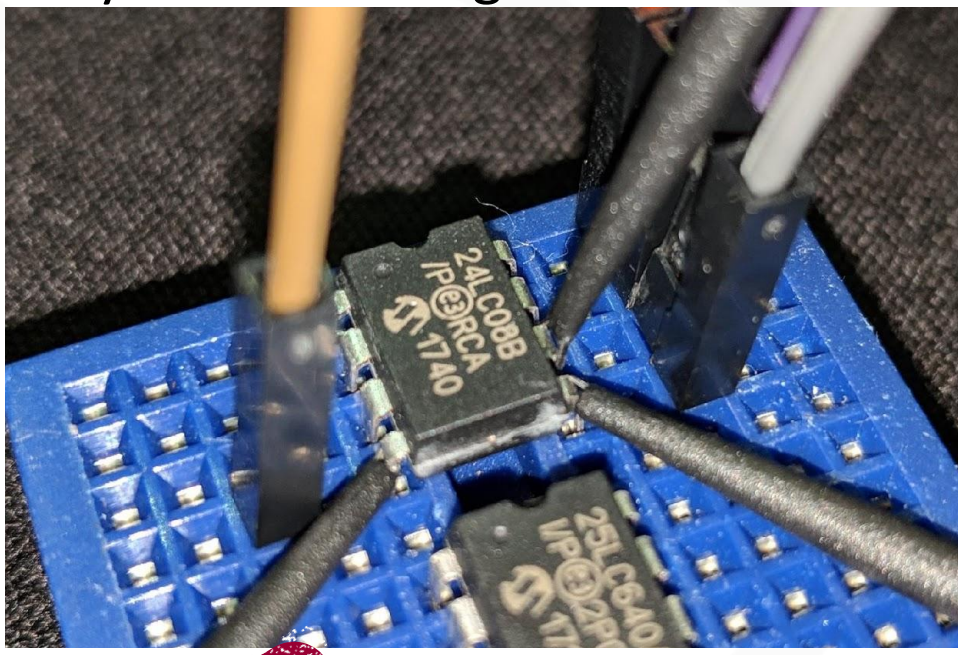
This is based on the now defunct Logic4, but should still apply to all modern Logic devices using their first four datalines.

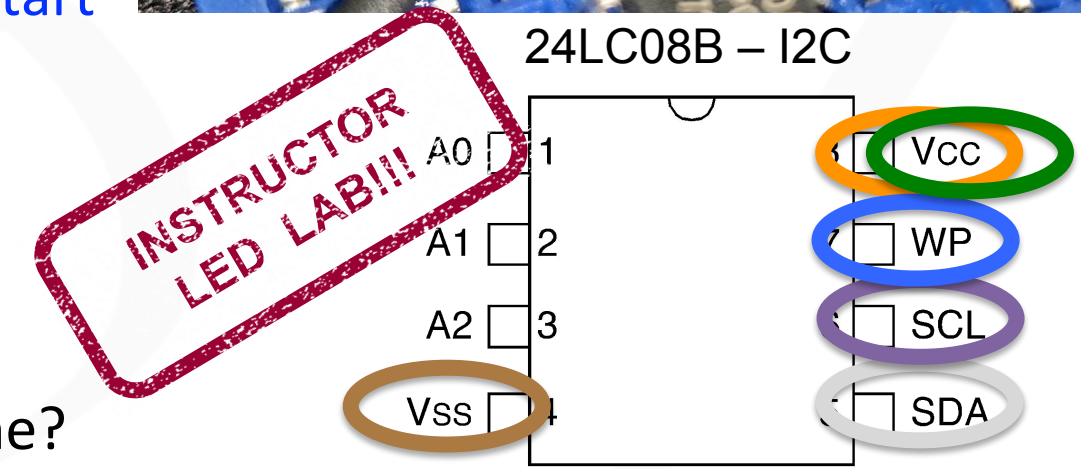
I²C Capture with Saleae Logic

- With your buspirate still connected to your EEPROM, wire your Saleae Logic4 device to your EEPROM
  - Probe 0 (black wire on top) to the SDA pin
  - Probe 1 (brown wire on top) to the SCL pin
  - Probe G (any of the black wires on bottom) to Vss/GND
- Connect your Saleae Logic4 to your Control Things
- Start the Saleae Logic Analyzer application
- On the right side of the screen, add an I²C analyzer
- Configure Saleae to record 60 sec at 3 MS/s and Start
- While capturing, run the i2c-dump script

```
i2c-dump -s 1024 -b 256 -o /tmp/24LC08B.bin
```

- Analyze by zooming into capture to verify success
- Export the Decoded Protocol to a csv file
- How many bytes did i2c-dump script read at a time?





SPI Capture with Saleae Logic

- With you buspriate still connected to your EEPROM, wire your Saleae Logic4 device to your EEPROM
  - Probe 0 to the MOSI/SI pin
  - Probe 1 to the MISO/SO pin
  - Probe 2 to the Clock/SCK pin
  - Probe 3 to the Enable/CS/SS pin
  - Probe G (under 0) to Vss/GND
- Open a terminal and type logic to start the Saleae Logic Analyzer
- Connect your Saleae Logic4 to Control Things VM
- Configure a SPI analyzer
- Configure Saleae to record 60 seconds at 12MS/s and start capture
- Capture the spi-dump script running

```
spi-dump -s 8192 -o /tmp/25LC640a.bin
```

- Analyze by zooming into capture to verify success
- Export the Decoded Protocol to a text or csv file and analyze