



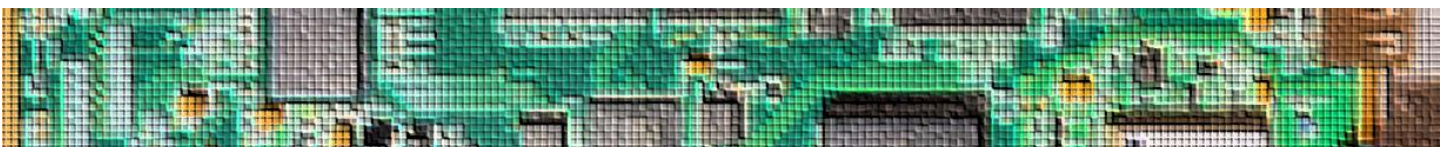
# Hardware Reverse Engineering

2023 Student Workbook

June 13, 2023

Instructor: Bill Hass

Source: <https://github.com/bhass1/cthwre.git>



# Table of Contents

<b>1</b>	<b>Welcome (~20 minutes)</b>	<b>3</b>
1.1	COVERED TOPICS	3
1.2	INTRODUCTION	3
1.3	BASIC TOOLS	7
1.4	ADVANCED TOOLS	8
1.5	RECOMMENDED BOOKS	9
<b>2</b>	<b>First Contact (~40 minutes)</b>	<b>10</b>
2.2	VISUAL SURVEY	10
2.3	CHIP IDENTIFICATION	12
2.4	PASSIVE PROBING	14
2.5	SECTION CHECKPOINT	14
<b>3</b>	<b>Hardware Modification (~30 minutes)</b>	<b>15</b>
3.1	BASIC SOLDERING	15
<b>4</b>	<b>With Power Comes Responsibility (~90 minutes)</b>	<b>17</b>
4.1	SECTION GOALS	17
4.2	MULTIMETER PROBING	17
4.3	OSCILLOSCOPE AND LOGIC ANALYZER PROBING	18
4.4	UART PROBING	20
4.5	ON-CHIP DEBUGGING - MEMORY READ/WRITE	24
<b>5</b>	<b>Extras</b>	<b>28</b>
5.1	SETUP THE SALEAE LOGIC ANALYZER	28
5.2	UPGRADE BUS PIRATE FIRMWARE	28
5.3	USING THE FTDI CABLE	30
5.4	BUS PIRATE UART PROBING (LEGACY)	30
5.5	BUS PIRATE WITH AVRDUDE FOR ICSP MEMORY READ/WRITE	33
5.6	JTAG WITH A BUS PIRATE	36
5.7	GLOSSARY	37

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA. It is attributed to Bill Hass and Russ Bielawski, and the original version can be found here: [https://github.com/downbeat/cthwre/blob/master/cybertruck\\_2017/hardware-reverse-engineering-students-2017.pdf](https://github.com/downbeat/cthwre/blob/master/cybertruck_2017/hardware-reverse-engineering-students-2017.pdf)

Modifications were made in 2019, 2021, and 2023 to update, add, and improve quality of content. Class structure has largely remained the same. 2019 & 2021 introduced new target hardware with new on-chip debugging lessons. 2023 introduced new software (tio & PulseView) and hardware (Tigard) tooling.

Copyright © 2023, 2021, 2019, 2017 Bill Hass; Copyright © 2017 Russ Bielawski

# 1 Welcome (~20 minutes)

## 1.1 Covered Topics

*What topics are covered?*

In this class, we will cover the following hardware topics:

- Basic usage of electronics tools:
  - o Multimeter
  - o Soldering iron
  - o TIGARD
  - o Oscilloscope
  - o Logic analyzer
  - o Serial interface device
  - o On-chip debugger & programmer
- Circuit identification – The process of reconstructing information about the circuitry on the printed-circuited board (PCB) of the target hardware to understand how it works.
- Board modification – The process of modifying the target hardware to enable breakout of interesting signals or change functionality to assist in hardware reverse engineering and exploitation. For this class, this means basic soldering.
- Serial data interfacing – Establishing serial data communication on the target hardware by tapping into UART pins for inspection, analysis and injection of commands or data.
- In-circuit debugging – Attaching to the built-in debugging circuitry of the target hardware to access memory and control the processor in real-time.

## 1.2 Introduction

*What is reverse engineering?*

Reverse engineering (RE) is an **iterative process** of discovery, planning, and experimentation to learn how something is designed, built, and used to achieve a function or goal. A reverse engineer aims to answer the what, why, and how about a completely unfamiliar system.

*What is hardware engineering?*

In computing, hardware engineering is an **iterative process** that involves designing and constructing the physical circuitry of an electrical system to perform a task or set of tasks. Hardware engineering is often carried out by electrical, mechanical, and/or computer engineers. Common tasks include: (1) functional block design; (2) active and passive component selection; (3) circuit design, layout, and routing; (4) circuit simulation; (5) board fabrication and rework; and (6) validation of assembled printed circuit boards (PCB).

### (1) Functional block design

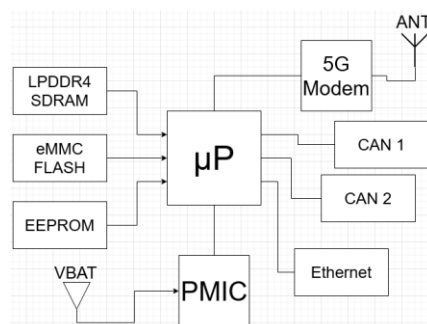


Figure 1: Example functional block diagram

## (2) Active and passive component selection (aka building a BOM)



Figure 2: Common electronics distributors

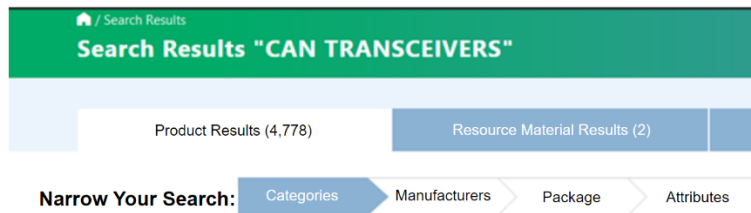


Figure 3: Search for CAN transceiver from electronics distributor website

## (3) Circuit design, layout, and routing

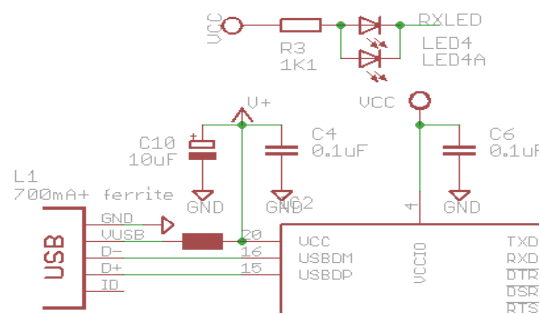


Figure 4: Example circuit schematic

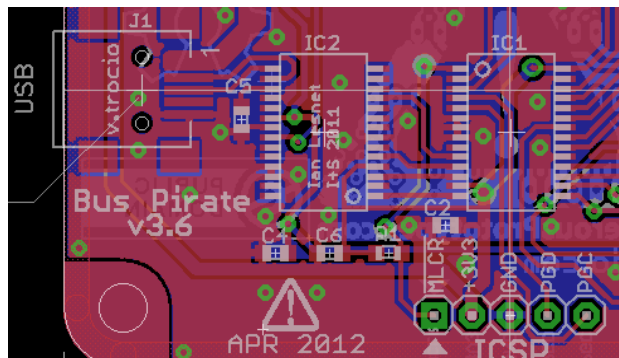


Figure 5: Example PCB assembly drawing

- (4) Circuit simulation - to check correctness, robustness, and other parameters of circuit design before costly & time-consuming fabrication.
- (5) Board fabrication and rework – initial fabrication is often done by dedicated facilities, but depending on quality, a HW eng. may need to “rework” the device by solder or desolder components or wires to correct errors or temporarily bypass components.

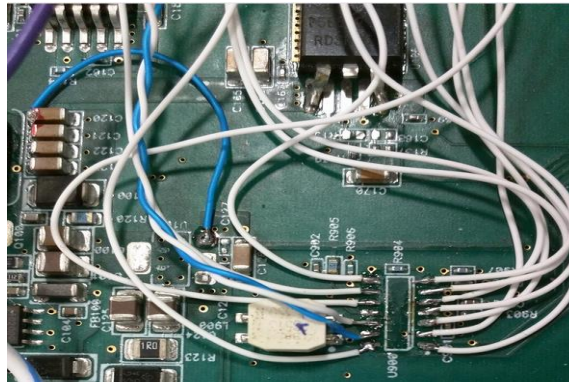


Figure 6: Example board rework

- (6) Validation of assembled PCBs - to ensure circuitry behaves properly before, during, and after different software stages are flashed (e.g. bootloader, OS, and applications). Involves powering up the device and verifying power domains are at the proper levels and loading test software to ensure chip communication channels are clean and correct.
- (7) Iterate! Spin a new revision of the device using the prior steps to fix any mistakes, add/replace components, etc.

#### What is HW RE?

Hardware reverse engineering (HW RE) is hardware engineering done in reverse. We iterate over these three steps:

(1) Discovery; (2) Planning; (3) Experimentation.

##### (1) Discovery - *information gathering and documentation*

- Reference searches are carried out (perhaps somebody has already reverse engineered your target device or something similar!)
- Physical components and markings are identified
- A component list, aka bill-of-materials (BOM) is created

Through discovery, the HW RE engineer knows about connectors, external memory chips, microcontrollers, external and internal network interfaces, and miscellaneous/benign components on the target.

##### (2) Planning - considers information from the discovery phase to *chart a path towards a goal, consider new goals, and prioritize next steps*

- From the discovery phase, you may have discovered hidden components or connectors that will make your job easier, or you may have noticed security features that dissuade you towards other low hanging fruit.

At the end of this phase, the HW RE engineer has a prioritized list of interesting things to try (a plan) and has a better feeling of what might work and what won't.

##### (3) Experimentation - *executes parts of the plan to achieve a goal, aids in discovering more information, and/or verifies assumptions from the prior phases.*

- This phase may involve powering up the unit for the first time and taking measurements.

After experimentation, the HW RE engineer might know what the different power domains are, what external and internal communication interfaces are active, and/or that a particular circuit or pinout is what they thought it was.

*Why do HW RE?*

A HW RE engineer provides crucial information for the rest of a comprehensive security assessment:

- **Analysis of vehicle networks:** Identification of “next-hop” devices (e.g. does the unit talk on any shared buses with safety or security critical modules?)
- **Locate and bypass physical security measures**
- **Identify key components:** Micros, memory, safety/security ICs, interfaces, & debug ports
- **Hardware engineering artifacts:** Pin diagrams, schematics, BOM, assembly drawings, etc.
- **Safe operation of target device:** Ability to power-up the unit without letting the smoke out
- **Runtime interaction w/ target device:** Reliable and robust connections to on-board interfaces
- **Target Device Software:** Bootloader, OS, configs, filesystems, binaries, libraries, etc.

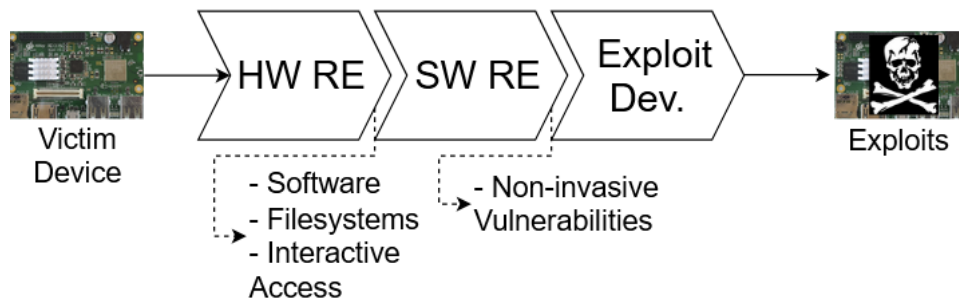


Figure 7: A common exploit development workflow: HW RE yields firmware; SW RE yields a vulnerability; and a vulnerability yields a (possibly remote) exploit.

This HW RE course is geared towards assisting the software reverse engineering (SW RE) process. SW RE is useful in offensive security assessments for finding and exploiting vulnerabilities and in defensive security assessments for finding and fixing vulnerabilities. To reverse engineer software for a device, however, the SW RE engineer needs access to the software. HW RE techniques can be used to obtain software from a target device in their possession (*It's usually better to search online for a software download first!*).

Embedded systems are frequently available physically to a reverse engineer (as opposed to, say, the hardware running a cloud service). By using hardware and software reverse engineering techniques, a reverse engineer can find "Very Bad Things." Like when there are unpatched vulnerabilities in one device's implementation that allow for remote exploitation on other devices resulting in a *scalable remote code exploit (RCE)*. This is surprisingly common due to design decisions like global keys, assumptions about physical security, and a general lack of adequate security features.

*What if I just want to hack hardware?*

Hunting for remote vulnerabilities is not the only reason to reverse engineer a piece of hardware. Local attacks alone are often attractive to owners seeking to achieve additional or altered functionality from a device they own. Owners of video game consoles modify the hardware and software of their consoles to allow for homebrewed games (and enable piracy), owners of automobiles use hardware and software modifications to “tune” their vehicles to achieve better performance or increase the value of their vehicle by rolling back the odometer, and farmers reverse engineer their heavy agriculture vehicles so they can diagnose and make their own repairs.



## 1.3 Basic Tools

To think about what tools might be useful, it's important to think about the kinds of tasks that might be useful when reverse engineering:

- Determining which points on a PCB connect to one another → Multimeter
- Modifying hardware to access signals or create new circuits → Soldering Station
- Analysis of analog signals → Oscilloscope
- Analysis of digital signals → Logic Analyzer
- Decoding of encoded data transmitted on digital signals → Oscilloscope or Logic Analyzer
- Injection of commands and/or data into hardware interfaces → Many, depends on interface (e.g. USB-to-Serial, USB-to-CAN, USB-to-LIN, USB-to-I2C, and USB-to-SPI adapters)
- Real-time analysis and control of the processors (or FPGAs, etc.) on the hardware → Many, depends on hardware (e.g. JTAG, In-Circuit Serial Programmer, Serial Wire Debug)

### 1.3.1 Multimeter

A multimeter is a tool which can measure AC/DC voltage, resistance, capacitance, AC frequency, and current. Most come with an auto-ranging function that automatically adjusts the order of magnitude of the measurement, but for those that don't you will need to make a rough estimate before setting its mode. Additionally, multimeters also have an extremely useful "continuity testing" mode that beeps when there is a short circuit between both probes. This functionality is particularly useful for mapping an unknown circuit because it lets you see if two points are short-circuited (directly connected) to one another. A lot can be learned about a circuit by using a multimeter, and I recommend this to be one of your first investments as a HW RE engineer.



### 1.3.2 Soldering station

Consisting of a soldering iron, flux, solder, a wet sponge or brass sponge, a desoldering pump or wick, a helping hands, spare wire, and a good light, a soldering station is essential for modifying and building hardware. New connections can be added (e.g. repopulate a header or tap onto an existing pad so probes can be attached) or existing connections can be removed (e.g. remove an external memory chip so it can be transferred to an external reader or disable a security circuit) greatly expanding the options available to a HW RE engineer. Along w/ a multimeter, a soldering station should be one of the first investments of a HW RE engineer.

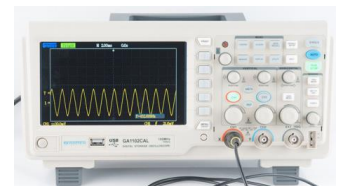
### 1.3.3 Tigard

The Tigard is a contemporary open-source hardware hacking multi-tool. <https://github.com/tigard-tools/tigard> has all the information you need. To summarize, it can perform serial data decoding and injection for several protocols including UART, SPI, I<sup>2</sup>C, JTAG, SWD, and ICSP. Some highlights are it's thoughtful hardware design, variety of pinouts, logic analyzer interface, and dedicated UART enabling UART to be used at the same time as other protocols.



### 1.3.4 Oscilloscope

An oscilloscope is a tool used for measuring analog signals in real-time. Different oscilloscopes will have different ranges of signal frequencies that they can measure, and faster oscilloscopes are (sometimes considerably) more expensive. The oscilloscope is a



great general purpose tool, and many oscilloscopes can also decode digital signals as well. Generally, however, once a digital signal has been identified and decoded with an oscilloscope, it is more useful to monitor with a logic analyzer or another digital decoding device.

### 1.3.5 Logic analyzer

A logic analyzer is a tool used to measure digital signals, and unlike oscilloscopes, most do not perform real-time monitoring. Instead, you set a trigger point and look at what was captured after the trigger fired. Further, they often require a PC to operate. With more channels and ability to capture longer waveforms than a similarly priced oscilloscope, logic analyzers are most useful when analyzing many channel digital protocols or characterizing FPGAs.



### 1.3.6 USB-to-serial adapter (a.k.a. FTDI cable)

A USB-to-serial adapter allows for sending & receiving data on serial buses using a PC. This specialized tool is often more useful than a logic analyzer alone because it allows the reverse engineer to interact with the serial port directly. These devices are sometimes called “FTDI cables,” because the company FTDI has a corner on the market of USB-to-serial adapter integrated circuits. Be aware that different voltage logic levels & PHYs exist. A 5V tolerant, 3.3V logic level TTL works in most cases, but sometimes RS-232, RS-485, or RS-422 PHY must be used.



### 1.3.7 Microcontroller-specific Debugger/Programmer

The Tigrad is a great Swiss Army Knife for a HW RE engineer’s toolkit, but it has its limitations. One limitation is that it can be slower than a specially designed debugger/programmer (e.g. 7 hours vs. 2 minutes to program a microcontroller). For this reason, it can be necessary to acquire a specialized programmer for the microcontroller you are working with. There are a few multi-purpose microcontroller-specific debuggers/programmers to choose from that have overlapping microcontroller support. Common devices include P&E Micro MultiLink, Segger J-Link, Lauterbach Trace32, and ATMEL AVR ISP.



### 1.3.8 Miscellaneous Parts

In addition to the major tools listed above, there are a number of tools a hardware reverse engineer is bound to need:

- Screwdrivers
- Razor blades
- Tweezers
- Pliers
- Strippers
- Q-tips
- Paper clips
- Scissors
- Electrical tape
- Hot glue gun
- DC power supply
  - Battery
  - Wall-wart
  - Bench-top
  - Adjustable
- Jumper wires
  - male-male
  - male-female
  - female-female
- Headers
- 30AWG wire wrap
- Mini grabber probes
- Prototype boards
- Linux computer
- USB A, B, C, mini, micro
- Magnifying glass
- A good light
- 30 AWG solder

## 1.4 Advanced Tools

The above tools are the most common tools general HW RE engineers will need, but depending on *your* goals and specific target, you may find yourself reaching for one of the advanced tools below. Besides the hot air station, the tools in this



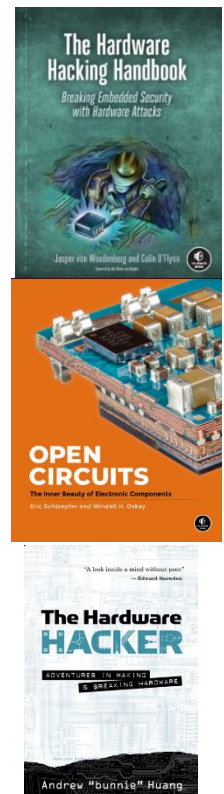
section will not be covered during the class. They are listed so you are aware that they exist and to provide pointers to where you can learn more about how they are used for HW RE.

1. **Hot Air Station** - A handheld hair-dryer-like device that blows hot air out of a small nozzle. Used to heat up areas of a board evenly which makes it easier to solder or desolder multiple pads/pins at once.  
<https://learn.sparkfun.com/tutorials/how-to-use-a-hot-air-rework-station>
2. **Solder Paste** - Tiny solder balls suspended in flux. Goes on like toothpaste. Makes surface mount soldering with a hot air station easier. <https://www.instructables.com/id/How-to-Surface-Mount-Solder-Using-Solder-Paste/>
3. **JTAGulator** - Nifty tool that automates checking test-points, vias, and pins for on-chip debug interfaces. Does so by enumerating possible pinouts. <http://www.grandideastudio.com/jtagulator/>
4. **Memory Sockets** - Mechanical contraptions that break out memory for you; used by memory suppliers to flash and test chips w/o a PCB. <https://www.digikey.com/products/en/development-boards-kits-programmers/programming-adapters-sockets/798>
5. **ChipWhisperer** - Nifty tool for side-channel analysis and glitching attacks.  
[https://wiki.newae.com/V5:Getting\\_Started](https://wiki.newae.com/V5:Getting_Started)
6. **3D Printer & C&C Mill** - Used to fabricate your own special purpose tools and jigs.  
<https://www.inventables.com/projects/pcb-milling-on-x-carve> <https://blog.adafruit.com/2017/06/01/3d-printed-pcb-workstation-with-needle-probes-by-giuseppe/>
7. **XRay** - Let's you see traces within PCB layers and other hardware secrets. <https://uvicrec.blogspot.com/2015/08/xy-ray-x-ray-scanner.html>
8. **Microscope** - Let's you look at decapped ICs. <https://seanriddle.com/decap.html>

## 1.5 Recommended Books

Knowledge is power.

1. **The Hardware Hacking Handbook: Breaking Embedded Security with Hardware Attacks**, by Jasper van Woudenberg and Colin O'Flynn – Published in 2021, it contains a few chapters covering similar content as this student workbook, but diving deeper and spanning broader than what we can fit here. Then, the bulk of the book discusses advanced hardware attacks like fault injection and side channel analysis in an approachable and enjoyable way. These professionals' perspectives are a must read!
2. **Open Circuits**, by Eric Schlaepfer and Windell H. Oskay – Published in 2022, the book is a visual & technical journey of a wide range of painstakingly cut, polished, and photographed electronic components from simple passives (resistors, capacitors, inductors, etc.) to complex integrated circuits. This book gives incredible insight with incredible detail to how components look within, which is helpful when you don't want to destroy the targets you are trying to reverse engineer and to help satisfy that curiosity itch you might get when looking at electronic hardware.
3. **The Hardware Hacker: Adventures in Making & Breaking Hardware**, by Andrew "bunnie" Huang – Published in 2019, this book gives a deep insight into the Chinese hardware markets in Shenzhen, brings you on a tour of different factories including where Arduinos are built, and shares both positive and negative aspects of hacking on hardware to bring a product to market.



## 2 First Contact (~40 minutes)

Before even powering up the hardware, look at the hardware itself. First contact with a new component often involves carefully taking apart the hardware or parts of the hardware to get to the circuitry to identify areas of interest. Be especially careful to avoid damaging the circuitry when attempting to physically separate components and enclosures.

### 2.1.1 Tools Used

- Screw drivers
- Razor blades
- Tweezers
- Pliers
- Scissors
- A good light
- Magnifying glass
- Multimeter

### 2.1.2 Section Goals

- 1) Note points of interest (Section 8) Locate microprocessor/microcontroller programming interface (Section 2.4 Passive Probing)
- 2) Visual Survey
- 3) Identify at least two interesting communication interfaces and how to connect to them (Section 8) Locate microprocessor/microcontroller programming interface (Section 2.4 Passive Probing)
- 4) Visual Survey
- 5) Create a BOM & assembly drawing (Section 2.3 Chip Identification)
- 6) Identify microprocessors/microcontrollers (Section 2.3 Chip Identification)
- 7) Identify power hookup and power domains (Section 2.4 Passive Probing)
- 8) Locate microprocessor/microcontroller programming interface (Section 2.4 Passive Probing)

## 2.2 Visual Survey

Once you have access to the electronics, take some time to look carefully at them to get an idea of what does what. Many times, the designers of the hardware use a layer of silk screen (printing on a PCB) to mark components with identifiers and even make comments on circuits. Even w/o silkscreen, you can learn a lot about a system just by looking at it.

- How are components mounted?

*Through-hole is your friend. Flat pack components expose all pins. BGA is your enemy.*

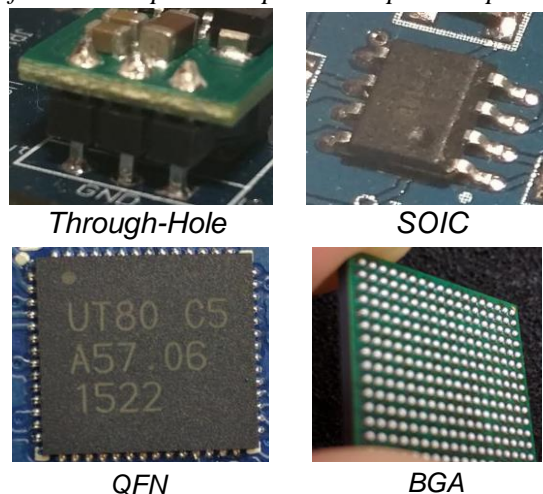
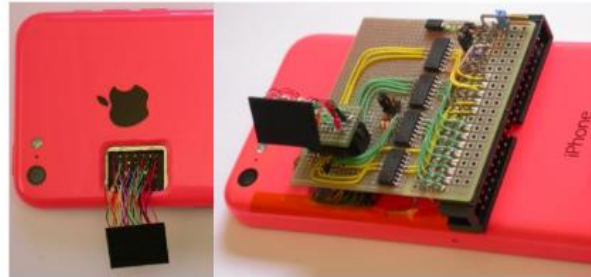


Figure 8: Example component packaging

*Advanced techniques can access BGA pads, but might not be worth your time..*



© Sergei Skorobogatov from  
"The bumpy road towards iPhone 5c NAND mirroring"

Figure 9: Working around BGA mounted components

- Are there barriers or protections in place?  
*EMF shielding, robustness coatings, and heat sinks can make our job difficult, but they can often be removed carefully.*
- What are the populated interfaces?  
*Things like USB, vehicle connectors, and hidden connectors.*
- Where are interesting areas (depopulated pads, test-points, unsure)?  
*Development and debug interfaces are typically depopulated before production, but they might still be supported by the software.*



Figure 10: Interesting areas of a target device

*A group of test points used by assembly line equipment might be used to flash software.*



Figure 11: Groups of test points on a PCB

- How do components relate to one another?  
*Observe general layout, components will be closest to what they interface with because that's cheaper and easier for the HW engineer.*
- How is the board powered?  
*You will eventually need to power the board. Good starting point for tracing. Big traces mean big current. There's a good chance the big trace is a power or ground line.*



Figure 12: Fat versus skinny traces on a PCB

- How many layers does the PCB have?

*A PCB with inner layers may have important communication buses embedded within, making our job harder to locate and tap into them.*

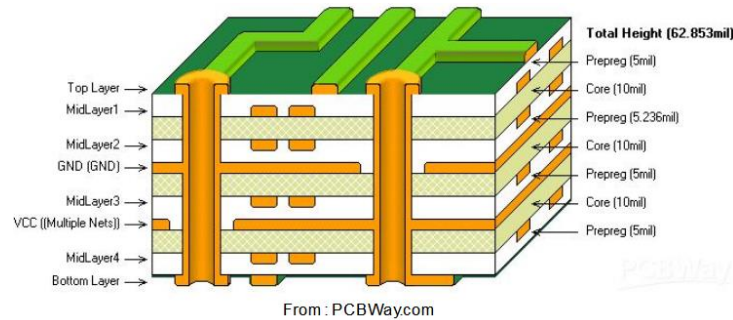


Figure 13: An 8-layer PCB

*Vias can be used to determine if there are inner layers.*

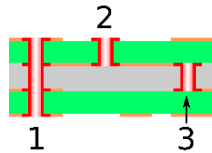


Figure 14: Types of vias; (1) via; (2) blind-via; (3) hidden-via

## 2.3 Chip Identification

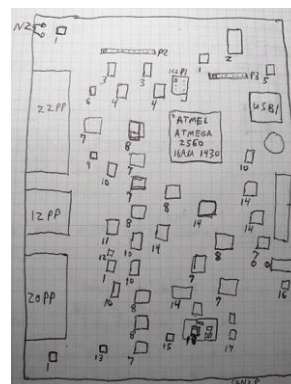
Gather information about each chip on the board, build a “Bill-of-Materials” (BOM) and PCB assembly map. Useful resources: <http://www.smdmark.com/en-US/>; <https://duckduckgo.com/>; <https://www.elnec.com/en/support/ic-logos/>, or any of the common electronics distributors in Section 1.2. Identify the major components on the target device and try to find their manuals on the internet.

- What are each of the chips and what do they do?

*Identify **every** chip on the board. Draw a diagram or take a picture and work through every component from top-left to bottom-right. Note processors, FPGAs, interface controllers, and memory. The more you know the better.*

```
-----
-- Main Micro --
-----
Microchip
ATMEL ATMEGA 2560
http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-
-----
--COMMS (USB, CAN, LIN)--
-----
2: CY7C 65213 28PVXI
Cypress
USB-UART LP Bridge Controller
https://www.cypress.com/file/139881/download
-----
3: MCP 2561E
Microchip
High-Speed CAN Transceiver
https://www.microchip.com/wwwproducts/en/MCP2561
```

BOM






























































































































































































## PCB Assembly Map

Figure 15: Example BOM and assembly map

## ELECTRONIC COMPONENT MANUFACTURER LOGOS

2015-01-06.002

	Abracon		Ericsson		Lattice Semi.		Omniel		Solid State Inc.
	Advanced Analogic		Exar		Lattice Semi.		ON Semiconductor		Solid State Scientific
	Advanced Linear Devices		Fairchild		Lattice Semi.		Optek		Solomon Systech
	Advanced Power Technology		Ferranti		LG Semi.		Pericom Semi.		STMicroelectronics
	Agilent		Fox Electronics		Linear Tech.		Plessey		Suryuan
	Alliance Semi.		Freescale		Littelfuse		Power Innovations		Synertek
	Allegro Microsystems		Fujitsu		Lucent		Power Integrations		Taiwan Semi.
	Alpha Semi.		Fujitsu		M Systems		Power Trends		TDK Semi.
	AMD		GEC Plessey		Macronix		Quality Semi.		TE Connectivity
	American Zettler		General Electric		Macronix		QuickLogic		Teccor
	Anadigics		General Semi		Marvell Semi.		Rabbit Semi.		Teccor
	Analog Devices		General Semi		Matsushita		Ramtron		TelCom Semi.
	Analog Devices		Gennum		Matsushita		Realtek		Teledyne
	Aries		Gennum		Maxim		Rectron		Texas Instruments
	Astec		Gould		Micrel		Reliance		Thomson
	Benchmarq		Harris		Micro Linear		Renesas		Toko America
	Bothhand		Harris		Microchip		Rockwell		Torex
	Bright Power		Hitachi		Micronas		Samsung		Toshiba
	Burr-Brown		Holtek		Micronix		Sanken		Toshiba
	Catalyst Semi.		Hyundai		Microsemi Corp.		Sanyo		Trident
	Catalyst Semi.		Hyundai		Microsemi Corp.		Seiko Epson		Tripath
	Ceramate		IC Works		Mitel		Seiko Instruments		Triquint Semi.
	Chrontel		icube		Mitsubishi		Semitron		Tseng Labs
	Cirrus Logic		Integrated Device Tech.		Monolithic Memories, Inc		Semitron		Tundra Semi.
	CML Microcircuits		Integrated Device Tech.		MOS Technology		Semitronic		UMC
	Coiltronics		Integrated Device Tech.		Mosaic Semi.		Semtech		Unitrode
	Conexant		Integrated Device Tech.		Mosel Vitelic		SGS		Unitrode
	C.P. Clare		Integrated Device Tech.		Mostek		Shindengen		Vantis
	CUI Inc.		Impala Linear		Motorola		Siemens		Vitalic
	Cygna		Infineon		Mullard		Sierra Semi.		VLSI Tech.
	Cypress		Immos		Murata		Signetics		VLSI Tech.
	Cypress		Intel		Murata		Silicon Labs		Winbond
	Daewoo		International Rectifier		National Semi.		Silicon Storage Technologies		Xicor
	Dallas Semi.		Intersil		National Semi.		Silicon Systems		Xilinx
	Datal-Intersil		Knowles Acoustics		National Semi.		Silicon Systems		Zilog
	EG&G Reticon		Kota		Nordic Semi.		Siliconix		Zilog
	E.M. Marin		Lambda Elect.		Nvidia		Simtek		Zilog
	E.M. Marin		Lattice Semi.						

This image is a quick reference showing the logos used to mark electronic components by many major manufacturers. Logos that contain the company name or initialism (three or more letters) are excluded.

Maintained by [KrisBlueNZ@gmail.com](mailto:KrisBlueNZ@gmail.com)

Latest version at <http://www.electronicpoint.com/resources/47>



## 2.4 Passive Probing

With the board unpowered still, reverse engineer the circuitry to better understand the board function and focus in to areas of interest. Draw schematics by hand as you develop an understanding during this phase. You will use the multimeter for this task. Put it in continuity test mode. You can find this mode on your multimeter by looking for the “Wi-Fi” symbol. See Figure 16 for an example.

Note that continuity testing applies a voltage to the circuit and measures the response. Therefore, it’s important the target is unpowered during continuity testing as to not interfere with the reading. Also note that there is a resistance threshold for “beeping,” so even if you hear a beep, check that the resistance value displayed is very close to 0 to confirm there is continuity. Figure 17 shows a basic circuit and expected continuity results.

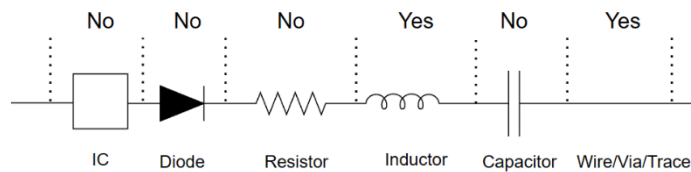


Figure 17: Example circuit showing continuity test results

- How are inputs and outputs connected to chips?  
*Identify passive circuitry, draw it out, reason about it.*
- What chips are connected?  
*Buses between memory and MCU or between interfaces and MCU could be MITMd.*
- Where do depopulated pads and test-points connect to?  
*This can help identify JTAG or serial interfaces and areas to be repopulated.*

## 2.5 Section Checkpoint

2.5.1 What is your device serial number? e.g. “S/N xxxxxxxx”	
2.5.2 What is your device hardware revision code? e.g. “Rev: ZF”	
2.5.3 What markings are on the main microcontroller?	
2.5.4 What package is the main microcontroller?	
2.5.5 What is the silkscreen code of the regulator powering the main microcontroller? e.g. “Uxxx”	
2.5.6 Where will you connect 12V power?	
2.5.7 What markings are on the quad UART?	
2.5.8 What is the silkscreen code of the main microcontroller JTAG interface? e.g. “Uxxx”	

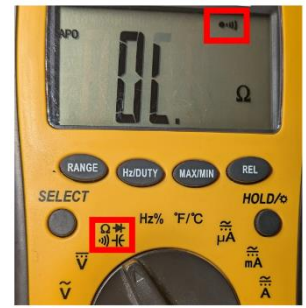


Figure 16: Close cropped view of multimeter configured for continuity testing

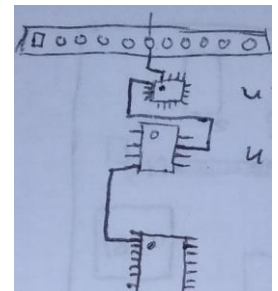


Figure 18: Example circuit drawing

### 3 Hardware Modification (~30 minutes)

#### Tools Used

- Soldering station
- Male header pins
- A good light
- Tweezers
- Magnifying glass
- Multimeter

#### Section Goals

- 1) Add male header pins to unpopulated interfaces of interest

### 3.1 Basic Soldering

#### 3.1.1 Concepts

- Solder flows toward heat & flux helps solder flow
- Heat dissipates quickly through copper & large contacts and planes (e.g. Vcc & Gnd) dissipate heat more quickly
- The second law of thermodynamics:
  - Over time, everything heats up to match the soldering iron temp
- A hotter iron will melt solder faster and damage sensitive components sooner
  - But too cool, and the whole circuit heats up before solder melts

#### 3.1.2 Best Practices

1. Wear safety glasses & DO NOT BREATHE IN THE FUMES!
2. Start with a hot iron (~250°C); if using flux, apply flux to target copper pad
3. When hot, clean the iron tip by applying solder then plunging it into the brass sponge. This is called “tinning” the tip and should appear shiny like the rightmost image below.



Figure 19: Three stages of cleaning a soldering iron tip.

4. Briefly heat target copper pad with clean, hot iron tip
5. Keep hot iron tip in place and push solder onto copper pad
6. Remove solder, then hot iron tip

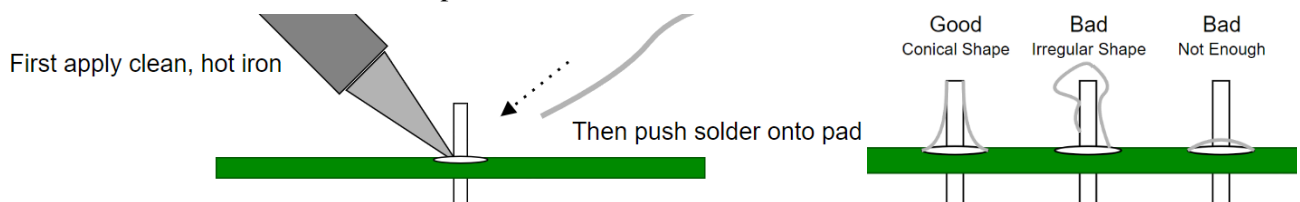


Figure 20: Basic steps of soldering through-hole components & quality of solder joints.

7. Observe quality of solder joint, fix with clean hot iron tip if needed.

***- Page Intentionally left blank. Use as scratch sheet. -***

## 4 With Power Comes Responsibility (~90 minutes)

After a thorough assessment with the board unpowered, it is time power it up. In industry, the first time you power the board is called “smoke-testing.” If you let out the smoke, your test fails because there’s no way to put it back in once it’s out.

**Note:** Before you power the board, check to make sure there aren’t any accidental short circuits!

In automotive, most of the time you will find 12v DC is the proper power supply voltage because a vehicle battery is 12v DC. However, this is not always the case. First, look for markings on the case or power supply voltage in the documentation. Then, if that doesn’t help and you are completely unsure, start at a lower voltage (around 5v) and gradually work up until the board appears to be functioning properly. Having a variable power supply helps a lot here, but there are other ways to test various voltage levels (e.g. use a USB power supply to get 5v). Luckily for you, automotive electronics are usually built with robust power circuits that can tolerate poor power conditions. So, as long as you are within a few volts and *have enough current* (and don’t have a short circuit), the system should function properly.

Multimeter probing at the different power domains will enable you to verify the board is powered properly.

### Tools Used

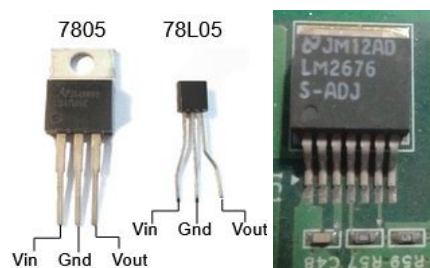
- Power Supply
- Multimeter
- Tigard
- Oscilloscope
- Logic Analyzer
- FTDI Cable
- Microcontroller Specific Debugger/Programmer
- Laptop/PC

### 4.1 Section Goals

1. Verify power domains and pin-outs (Section 4.2 Multimeter Probing & 4.3 Oscilloscope and Logic Analyzer Probing)
2. Establish communication with serial port of main micro (Section 4.4 UART Probing)
3. Dump main micro firmware (Section 4.5 On-Chip Debugging - Memory Read/Write)
4. Write modified firmware (Section 4.5 On-Chip Debugging - Memory Read/Write)

### 4.2 Multimeter Probing

Start at the power input connector and use the multimeter to check voltage levels at various points on the board. Focus on finding and measuring voltage regulators first, then measure voltage levels at the microprocessor. **BE CAREFUL NOT TO SHORT-CIRCUIT ANYTHING BY ACCIDENT!**



Examples of voltage regulators

- Verify the voltage domains.

*Know voltage domains to interface with board later without creating smoke (e.g. whether to use a 3.3V UART vs. 5V UART).*

- Use to verify chip and connector pin-outs.

*Sanity check what you think you know about the components. Are GND and Vdd where you expect them to be? Are they at the correct voltages based on their specs?*

**Tip:** A multimeter can also be used to find or verify digital network buses (e.g. UART or CAN).

*How? The DC voltage on a digital network bus has a steady state voltage and will fluctuate rapidly while data is transmitted. Power up the board and monitor the DC voltage on a suspected digital bus.*

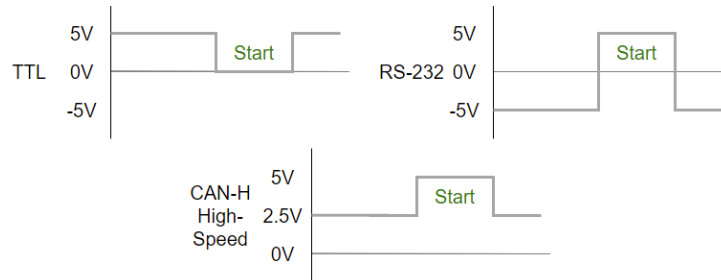


Figure 21: Steady-state and start bits of common digital communication protocols

#### 4.2.1 Multimeter Probing Checkpoint Questions

4.2.1.1. What is the measured output voltage of that regulator? This is the input voltage of the LH7A400.	
4.2.1.2. Based on the LH7A400 spec, there are two "Operating Voltages." Which one did you just measure?	

### 4.3 Oscilloscope and Logic Analyzer Probing

Probing with an oscilloscope or logic analyzer will give you a better picture of what is happening on the wire. Recall that an oscilloscope measures and displays an analog waveform while the logic analyzer measures and displays a digital waveform.

- Find digital network buses.

*Although a multimeter can help with this, it is much better to use an oscilloscope or logic analyzer here. Directly monitor various test-points and pins you suspect to be a digital network bus so you can see if they're active. Section 4.4 UART Probing will walk you through this.*

- Determine baud rates.

*When an asynchronous digital network bus (e.g. CAN or UART) is found, the next question is usually: "What baud rate is it at?" To measure, look at the time spent during the **shortest** bit pattern you can see. This is called the bit time. Simply invert the bit time to get the baud rate (e.g.  $(8.68\mu s)^{-1} = \sim 115200\text{bps}$ ).*

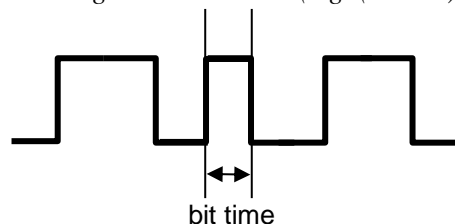


Figure 22: Measuring bit time to calculate baud rate

- Decode signals

*Logic analyzers and most oscilloscopes are able to decode signals in real-time. This information could be useful.*

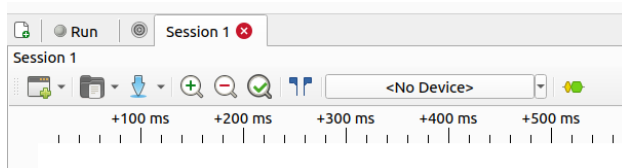




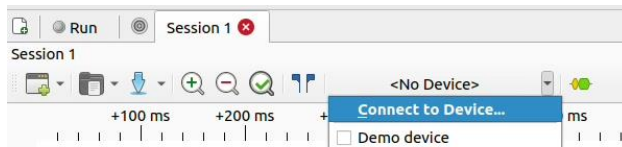
Figure 23: Oscilloscope decoding TTL UART waveform representing the character 'a'

### 4.3.1 Setup the BitMagic Basic Logic Analyzer

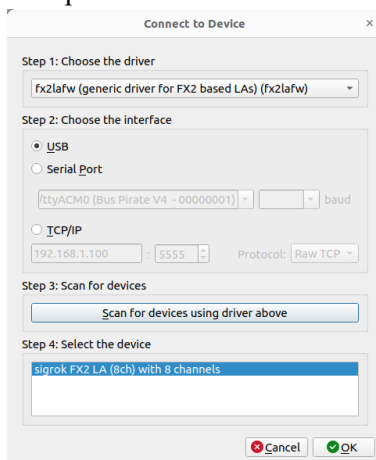
- Download and install the latest SigRok + PulseView from the official site <https://sigrok.org/wiki/Downloads> (Latest version was 0.4.2 as of 05/24/2023).
  - For Debian (Ubuntu) systems, `sudo apt install sigrok` works fine. Note this was probably already done on your lab machines. Try `pulseview` from your shell.
- Open the “PulseView” GUI application either from the app drawer or by typing `pulseview` in the terminal. You should see a similar screen.



- Connect to Device...



- Select the “fx2lafw” driver; USB interface, then scan for devices. Select “sigrok FX2 LA (8ch)” from the list, then press “OK.”



- Now, you are ready to collect some data.

## 4.4 UART Probing

UART stands for “Universal Asynchronous Receiver/Transmitter.” It is an old and low-cost serial protocol that is very common in embedded devices today. Often, it can be used by reverse engineers to do some pretty awesome stuff if it isn’t locked down:

- Obtain an interactive serial console with a target platform
- Read system and debugging logs
- Upload custom software
- Modify configuration parameters
- Bypass security features
- Dump memory
- Spoof embedded components

SparkFun has a great write-up on UART here: <https://learn.sparkfun.com/tutorials/serial-communication>.

### 4.4.1 Section Goals

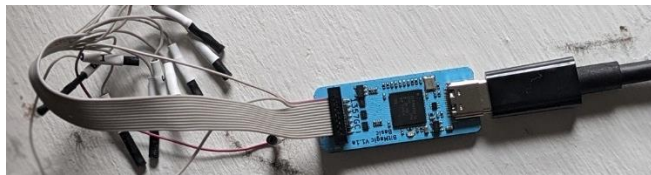
1. Establish communication with serial port of main micro

### 4.4.2 UART Rx

We will leverage our logic analyzer capabilities to get a dump of the UART communication happening on the target. This is nice because we can parallelize our analysis by capturing many UART channels at once, enabling us to identify which UART is more interesting so we can hone in on that one.

Two downsides are: 1) you can’t write back to UART with the logic analyzer & 2) logic analyzers aren’t meant to capture indefinitely. If you already know the UART Rx & Tx you are most interested in, skip ahead to Section 4.4.3 UART Tx. If not, read on...

We will use the BitMagic Basic with the flywire pigtail.



*Figure 24: Close cropped view of BitMagic Basic with flywire pigtail*

We’ll also use mini grabber test clips to attach to various pins and enable us to interface with the components non-invasively, without requiring a solder joint. This lets us quickly attach and reattach to different points of interest on the target.



*Figure 25: Close up of a mini grabber test clip (left); detail of open mini grabber claw (right)*

- Wire BitMagic Basic Logic Analyzer probes to your target. See the block diagram in Figure 26. You don't need RTS and CTS for UART, but they can be helpful. Don't wire VCC. Do wire GND.

- Note:** You are supposed to figure this part out yourself! You should know from the previous stages what and where to connect. If not, check your work and come back 😊

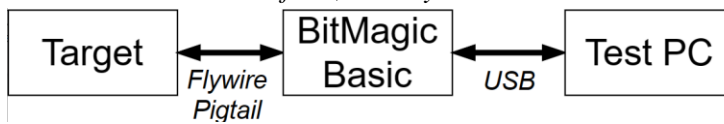
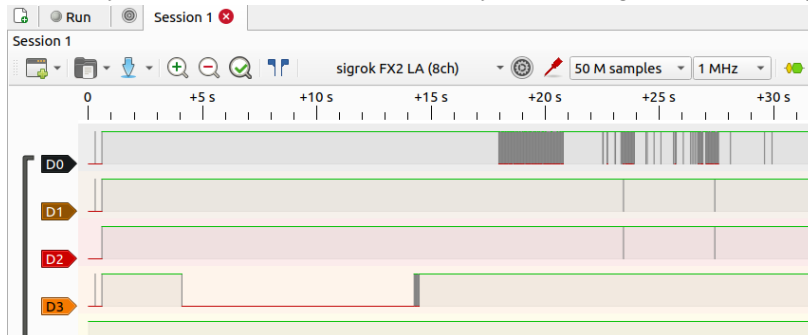


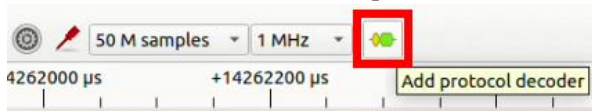
Figure 26: Block diagram for UART Rx using BitMagic Basic Logic Analyzer

- Hint:** If you connect to all 4 UART Tx, you should get a similar output as below.

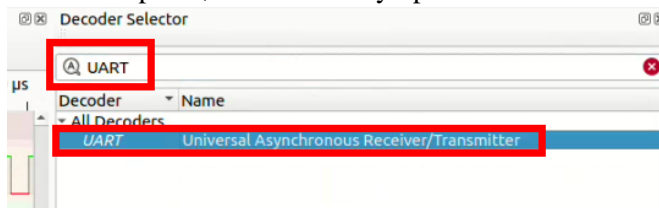


- Add a decoder for UART for a channel of your choice. You can add more than one.

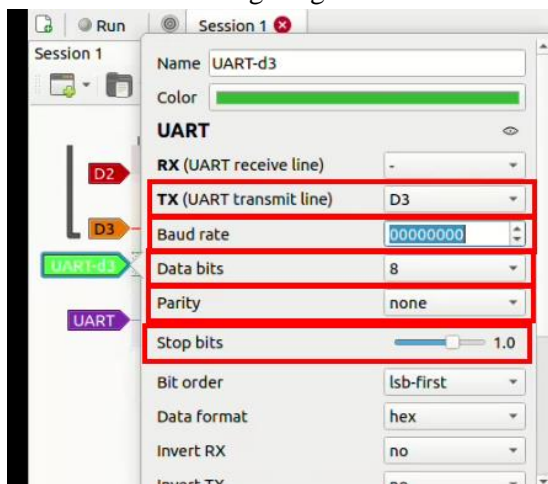
- Find the toolbar icon for “Add protocol decoder” and click it.



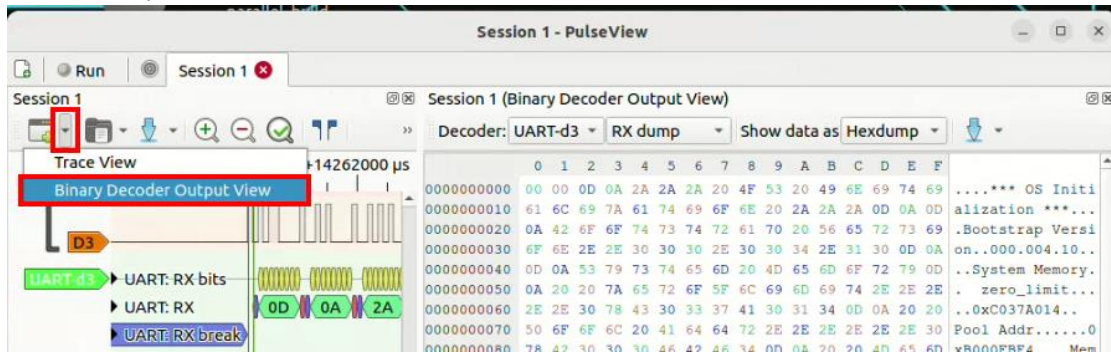
- In the new panel, there are many options available. Filter and select UART.



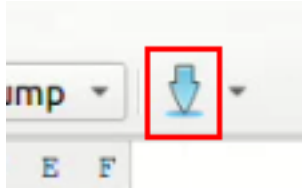
- Configure the UART parameters for the UART protocol decoder by left-clicking then filling the information. The most common configuration fields are outlined in red below. Use what you learned in Section 4.3 for configuring the baudrate.



3. Add a binary view for the decoder.



4. The decoder output view is cramped, and can be difficult to use for in-depth analysis. So dump the binary output of the capture for an easier analysis.



5. Use UNIX tools like ``strings``, ``cat``, ``xxd``, ``less``, and/or ``grep`` to analyze the data.

#### 4.4.2.1 UART Rx Checkpoint Questions

4.4.2.1.1 What is the “Trace Header type” set to?	
4.4.2.1.2. What is the Mobile Equipment Identifier (MEID) of your target?	
4.4.2.1.3. What is the International Mobile Subscriber Identity (IMSI) of your target?	

#### 4.4.3 UART Tx

It can be easier to use a dedicated single-purpose tool like the USB-to-UART cable (aka “FTDI cable”) as shown below. There are different types of cables supporting dedicated voltage levels (commonly 1.8V, 3.3V, or 5.0V) and dedicated PHY protocols (commonly TTL, RS232, or RS485). But those cables can add up quickly in terms of cost and space.

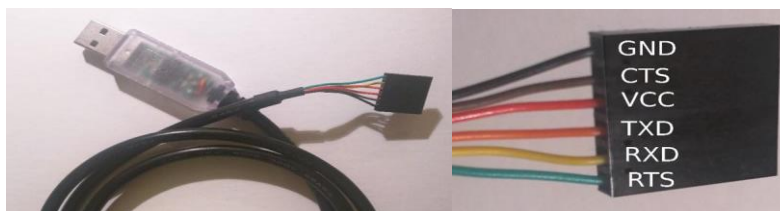


Figure 27: FTDI cable for UART communication (left); Pin-out of FTDI cable (right)

If you have a Tigard on hand, you can use that instead! We will be using the Tigard and BitMagic Basic for this section setup as shown in Figure 28 below.

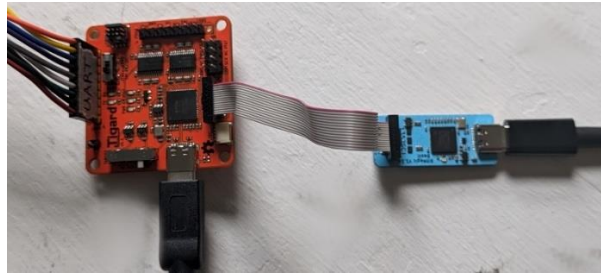


Figure 28: Close cropped view of Tigard with UART flywires connected to BitMagic Basic

1. Connect the Tigard USB-C to your PC and check that your OS recognizes it: ``sudo dmesg``

```
886.578958] usb 3-4.3: USB disconnect, device number 12
898.064674] usb 3-4.3: new high-speed USB device number 13 using xhci_hcd
898.169362] usb 3-4.3: New USB device found, idVendor=0403, idProduct=6010, bcdDevice= 7.00
898.169372] usb 3-4.3: New USB device strings: Mfr=1, Product=2, SerialNumber=3
898.169376] usb 3-4.3: Product: Tigard V1.1
898.169380] usb 3-4.3: Manufacturer: SecuringHardware.com
898.169382] usb 3-4.3: SerialNumber: TG11061d
898.171291] ftdi_sio 3-4.3:1.0: FTDI USB Serial Device converter detected
898.171333] usb 3-4.3: Detected FT2232H
898.171550] usb 3-4.3: FTDI USB Serial Device converter now attached to ttyUSB0
898.174770] ftdi_sio 3-4.3:1.1: FTDI USB Serial Device converter detected
898.174804] usb 3-4.3: Detected FT2232H
898.174936] usb 3-4.3: FTDI USB Serial Device converter now attached to ttyUSB1
```

2. List serial devices with ``tio --list-devices`` then connect to the Tigard's first serial interface with ``tio <tty-device> -b <baudrate>``

```
dev1@dev1-desktop:~$ tio --list-devices
/dev/serial/by-id/usb-SecuringHardware.com_Tigard_V1.1_TG11061d-if00-port0
/dev/serial/by-id/usb-SecuringHardware.com_Tigard_V1.1_TG11061d-if01-port0
dev1@dev1-desktop:~$ tio /dev/serial/by-id/usb-SecuringHardware.com_Tigard_V1.1_TG11061d-if00-port0
[22:18:25.407] tio v2.5
[22:18:25.407] Press ctrl-t q to quit
[22:18:25.408] Connected
```

- a. **Note:** You must set the baudrate with ``-b`` flag when connecting if you want anything other than default 115200. Set baudrate based on your findings from prior sections.
3. Power off your target, then connect the Tigard UART cable to the target. See Figure 29 for a block diagram.

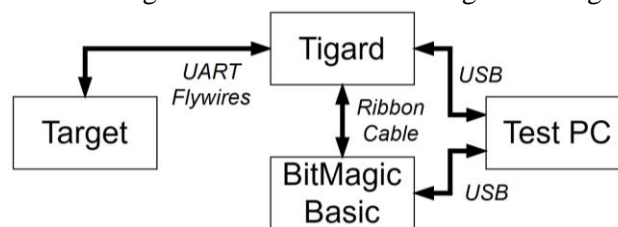


Figure 29: Block diagram for UART Tx using Tigard with BitMagic Basic

4. Switch the Tigard TARGET voltage switch to the correct voltage. **Hint:** It should \*not\* be set to VTGT unless you also have Tigard's UART VTGT connected to your target. I recommend keeping VTGT disconnected here. Check your datasheets and/or use an oscilloscope for voltage ratings to help you select this value.

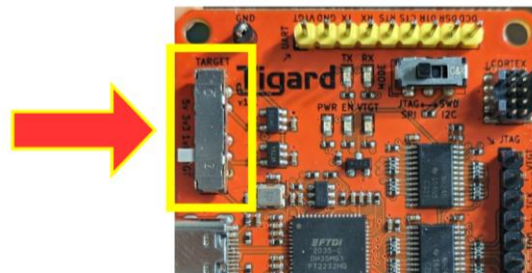


Figure 30: Close cropped view of Tigard's TARGET voltage switch



5. Setup PulseView as described in Section 4.4.2 so you can monitor the Rx & Tx lines of the Tigard.
6. Power on your target and watch tio for any output.
7. Use your keyboard to type characters into your tio terminal to transmit them on the Tigard's UART Tx line.
8. Use your PulseView skills to debug potential issues with your setup.

#### 4.4.4 Helpful tio Commands

In-Session Command	In-Session Key Combo
List available key commands	CTRL+t ?
Toggle local echo mode	CTRL+t e
Toggle line timestamp mode	CTRL+t t
Quit tio	CTRL+t q

CLI Option	CLI Flag
Display help	-h, --help
List devices	-L, --list-devices
Set baudrate (default:115200)	-b, --baudrate <baudrate>
Enable log to file	-l, --log
Enable line timestamp	-t, --timestamp

### 4.5 On-Chip Debugging - Memory Read/Write

On-chip debugging is crucial for development of complex embedded systems because it enables PCB-level and chip-level testing to diagnose and isolate potential failures in the hardware and software. It allows you to control the chips on the board at the lowest levels. Some example uses are:

- Write and read internal and external memory
  - o Extract firmware, modify it, write it back
  - o Modify configuration parameters
- Write and read internal registers and fuses
- Bypass security features
- Obtain an interactive debugging session with a program on the target platform

The microcontroller on the PeopleNet G3 OBC uses a 32-bit ARM9TDMI-family ARM922T processor according to the datasheet. Digging deeper (e.g. via Wikipedia) we can learn ARM922T uses the ARMv4T microarchitecture. Equipped with that information, we know we need to look for a JTAG port to begin on-chip debugging. JTAG stands for Joint-Test Action Group and is a widely used standard for on-chip debugging from 1990 by IEEE (IEEE Standard 1149.1). Additionally, the open-source ecosystem has mature support for the ARM architecture. Because of this, we can use low-cost tools to attach to the PeopleNet G3 OBC's JTAG port.

It is often easiest ("it just works") to use a microcontroller specific on-chip debugger and programmer as shown below, but they can be a couple hundred to a couple thousand dollars and only support a handful of microprocessors.



Figure 31: Examples of microcontroller specific debuggers and programmers: AVRISP mkII, Multilink, J-Link, TRACE32

As in Section 4.4 UART Probing, we will be using the Tigard here instead!

### 4.5.1 Section Goals

1. Bypass JTAG security (if you haven't done so yet)
2. Dump device firmware
3. Modify firmware and write it back

### 4.5.2 Setup

We will be using the same Tigard and BitMagic Basic setup for this section setup as we did in Section 4.4.3 UART Tx shown in Figure 28 above. This time, the Tigard will be our JTAG interface device and OpenOCD will be our interface software. OpenOCD, once setup, will use the Tigard as a protocol bridge to the ARM922T core running inside the NXP LH7A400.

1. Check that OpenOCD is installed with ``openocd -v``

```
dev1@dev1-desktop:~$ openocd -v
Open On-Chip Debugger 0.11.0
Licensed under GNU GPL v2
For bug reports, read
http://openocd.org/doc/doxygen/bugs.html
```

- a. **Note:** If needed, install OpenOCD from Ubuntu's package manager, APT with ``sudo apt install openocd``
2. Connect the Tigard USB-C to your PC and check that your OS recognizes it: ``sudo dmesg``

```
886.578958] usb 3-4.3: USB disconnect, device number 12
898.064674] usb 3-4.3: new high-speed USB device number 13 using xhci_hcd
898.169362] usb 3-4.3: New USB device found, idVendor=0403, idProduct=6010, bcdDevice= 7.00
898.169372] usb 3-4.3: New USB device strings: Mfr=1, Product=2, SerialNumber=3
898.169376] usb 3-4.3: Product: Tigard V1.1
898.169380] usb 3-4.3: Manufacturer: SecuringHardware.com
898.169382] usb 3-4.3: SerialNumber: TG11061d
898.171291] ftdi_sio 3-4.3:1.0: FTDI USB Serial Device converter detected
898.171333] usb 3-4.3: Detected FT2232H
898.171550] usb 3-4.3: FTDI USB Serial Device converter now attached to ttyUSB0
898.174770] ftdi_sio 3-4.3:1.1: FTDI USB Serial Device converter detected
898.174804] usb 3-4.3: Detected FT2232H
898.174936] usb 3-4.3: FTDI USB Serial Device converter now attached to ttyUSB1
```

3. Set the MODE switch of the Tigard to "JTAG SPI"



4. Set the TARGET switch of the Tigard to "VTGT" if you'll connect VTGT to your target (I recommend doing that here). Otherwise set it appropriately according to measurements or datasheets for interfacing with the main microcontroller. **Note:** This could be different from the voltage used in the UART sections because it's a different interface.



5. Power off your target, then connect the Tigard JTAG flywires to the target. See Figure 32 for a block diagram.

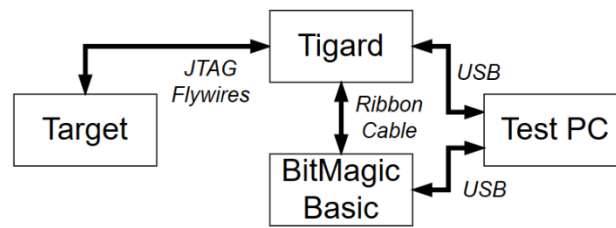


Figure 32: Block diagram for JTAG using Tigard with BitMagic Basic

6. Power on your target, then connect to it with OpenOCD via the Tigard interface tool. ``openocd -f tigard-jtag.cfg -f peoplenetg3obc.cfg``

```

dev1@dev1-desktop:~/source/cthwre/cybertruck_2023/students$ openocd -f tigard-jtag.cfg -f peoplenetg3obc.cfg
Open On-Chip Debugger 0.11.0
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
jtag
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : clock speed 2000 kHz
Info : JTAG tap: lh7a400.cpu tap/device found: 0x00922f0f (mfg: 0x787 (<unknown>), part: 0x0922, ver: 0x0)
Info : Embedded ICE version 2
Info : lh7a400.cpu: hardware has 2 breakpoint/watchpoint units
Info : starting gdb server for lh7a400.cpu on 3333
Info : Listening on port 3333 for gdb connections
  
```

### 4.5.3 Defeating JTAG Hardware Security

Did you receive the same message as above and successfully connect to the JTAG interface of the lh7a400? If not, check the datasheet of the lh7a400 and identify any potential hardware configuration that might need to be set for JTAG.

**Hint:** Take a closer look at the silkscreen for “MFG TEST” and re-read the paragraph above.

### 4.5.4 Dumping Flash

1. With OpenOCD connected to the target via Tigard interface, you can connect to it with telnet.
  - a. Open a new terminal and connect with telnet ``telnet localhost 4444``  
**Hint:** To exit telnet, use the key combination: ``Ctrl+] q``

```

dev1@dev1-desktop:~/source/cthwre/cybertruck_2023/students$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
  
```

- b. In the OpenOCD terminal, you should see the connection message

```

Info : accepting 'telnet' connection on tcp/4444
target halted in ARM state due to debug-request, current mode: Supervisor
cpsr: 0x200000d3 pc: 0x003a9184
MMU: disabled, D-Cache: disabled, I-Cache: enabled
  
```

2. In the telnet terminal, try listing the targets with ``targets`` then halting it with ``halt``. Listing with ``targets`` again should show the target in a halted state.

```
> targets
  TargetName      Type      Endian TapName      State
-----
0* lh7a400.cpu    arm920t    little lh7a400.cpu    running

> halt
target halted in ARM state due to debug-request, current mode: Supervisor
cpsr: 0x200000d3 pc: 0x003a9184
MMU: disabled, D-Cache: disabled, I-Cache: enabled
> targets
  TargetName      Type      Endian TapName      State
-----
0* lh7a400.cpu    arm920t    little lh7a400.cpu    halted
```

- With the CPU halted, let's see if we can dump some of the flash! ``dump_image test_dump_lh7a400.bin 0x0 0x1000``

```
> targets
  TargetName      Type      Endian TapName      State
-----
0* lh7a400.cpu    arm920t    little lh7a400.cpu    halted

> dump_image test_dump_lh7a400.bin 0x0 0x1000
dumped 4096 bytes in 0.074713s (53.538 KiB/s)
```

**Hint:** You can dump the entire contents of the flash this way. It is stored in the same directory as where OpenOCD was executed from.

- Now analyze the flash using UNIX tools for software reverse engineering like ``xxd``, ``strings``, and ``binwalk``.

#### 4.5.5 Writing Flash

Let's change part of the firmware and upload it back.

- Any strings that appear in the UART boot console are good candidates because we can ensure our changes really did take affect. ``vim test_dump_lh7a400.bin``
- Our flash won't respond to "CFI" driver commands unless it has been reset, so issue a ``reset init``
- When writing to NOR flash like the one present on this target, we need to use special subcommands in OpenOCD because it needs to be first erased before writing. List the flash banks on the board with ``flash banks``
- Some flash banks have write protection. Check this with ``flash info 0``
  - You may need to disable flash protection ``flash protect 0 <first> <last> off`` where `<first>` and `<last>` is the range of protected blocks you want to configure.
- Now write the modified flash binary to flash! ``flash write_bank 0 test_dump_lh7a400.bin``

## 5 Extras

### 5.1 Setup the Saleae Logic Analyzer

1. Download and install the latest Saleae Logic software from the official site: <https://www.saleae.com/downloads/> (Latest was version 2.3.33 at the time of writing)
2. Open the installed application and plug in the Saleae device into your PC's USB port. You should be greeted with a similar screen as below:

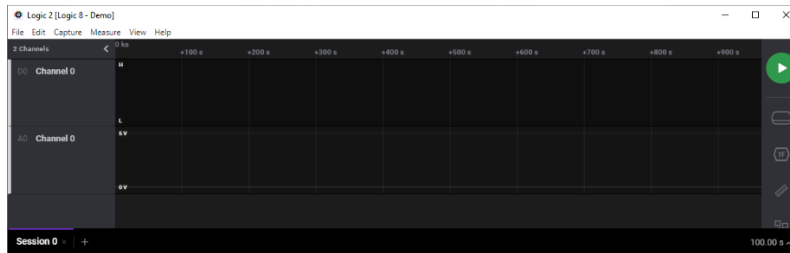


Figure 33: Saleae Logic 2 start screen

3. Configure the logic analyzer by clicking the icon below the big green play button. Select the channels you are interested in. I have 0, 1, and 3 attached to my target device. Also set the sample rate to something reasonable. A large sample rate and long capture will create a lot of data, so find a balance between the settings and storage space available. Below, I used 10MS/s and 1 second or more depending on what I am capturing. If you want, set up a trigger on one of the channels to begin capturing when the voltage on one of the channels changes.

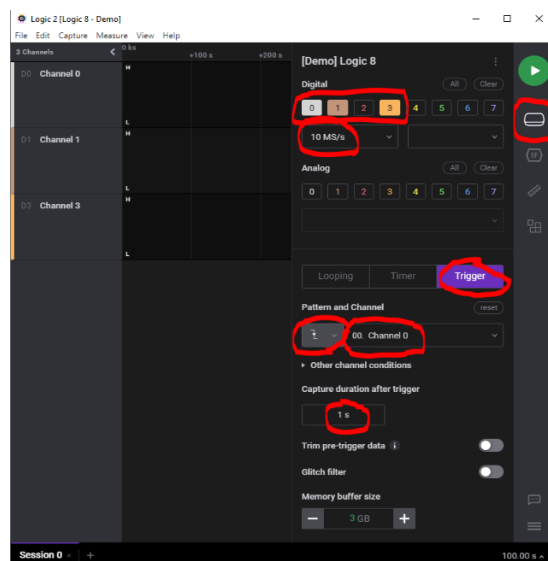


Figure 34: Configuring your device in Saleae Logic 2.

4. Press the big green play button to begin the capture.

### 5.2 Upgrade Bus Pirate Firmware

Older Bus Pirate firmware versions do not support all of the features we will be using. Therefore, you must upgrade the Bus Pirate firmware. I have provided a custom firmware for the CyberTruck HW RE (CTHWRE) based on v7.1 community edition.



- Download the CTHWRE Bus Pirate firmware and updater script from the public git repo and place them in the same folder.

[https://github.com/bhass1/cthwre/cybertruck\\_2019/tools/buspirate/bpv4\\_cthwre\\_firmware\\_final.hex](https://github.com/bhass1/cthwre/cybertruck_2019/tools/buspirate/bpv4_cthwre_firmware_final.hex)

[https://github.com/bhass1/cthwre/cybertruck\\_2019/tools/buspirate/pirate-loader\\_lnx](https://github.com/bhass1/cthwre/cybertruck_2019/tools/buspirate/pirate-loader_lnx)

**Note:** For intrepid readers at home, you can build the latest firmware from the official Bus Pirate GitHub:

[https://github.com/BusPirate/Bus\\_Pirate/blob/master/Documentation/building-and-flashing-firmware.md](https://github.com/BusPirate/Bus_Pirate/blob/master/Documentation/building-and-flashing-firmware.md).

**Note:** You'll also need to build the Bus Pirate v4.0 pirate-

[https://github.com/bhass1/cthwre/CyberTruck\\_2019/bpv4\\_cthwre\\_firmware\\_final.hexoader](https://github.com/bhass1/cthwre/CyberTruck_2019/bpv4_cthwre_firmware_final.hexoader) if you are using a Bus Pirate v4.0. You can find the source and build-script from the official Bus Pirate GitHub here:

[https://github.com/BusPirate/Bus\\_Pirate/tree/master/package/BPv4-firmware/pirate-loader-v4-source](https://github.com/BusPirate/Bus_Pirate/tree/master/package/BPv4-firmware/pirate-loader-v4-source).

- Connect to the Bus Pirate to the USB port of the PC
- Put the Bus Pirate into bootloader mode:
  - o **Bus Pirate v3.6**
    - Give permissions to /dev/ttyUSBx and connect to the Bus Pirate via minicom:
 

```
sudo chmod o+rw /dev/ttyUSBx
minicom -b 115200 -D /dev/ttyUSBx
```
    - If the Bus Pirate prompt doesn't appear, hit enter or press '?' (help) or 'i' (version information) and hit enter and the prompt should appear (possibly after some other output).
    - At the Bus Pirate prompt in screen press '\$' and, when prompted, 'y'.
 

```
HiZ>$
Are you sure? y
BOOTLOADER
```
  - o **Bus Pirate 4.0**
    - Short the PGC and PGD pins on the ICSP header. *Keep them shorted for the remaining steps.* You can solder a male header and use a female-female jumper wire or use a male-male jumper wire and wedge it into the through-holes.
    - While PGC and PGD pins are shorted, press the RESET button on the BPv4.
- Finally, we're ready to update the firmware in the Bus Pirate. Open a terminal (if one isn't already open) and change the directory to where the two files in the first step where downloaded.

```
cd ~/bp_loader
```

- Set the pirate-loader\_lnx\_bpv4 file we downloaded to executable and run it (both as root):

```
sudo chmod +x pirate-loader_lnx_bpv4
```

```
sudo chmod o+rw /dev/ttyACM0
```

```
./pirate-loader_lnx_bpv4 --dev=/dev/ttyACM0 --hex=bpv4_firmware_cthwre.hex
```

```
$ cd ~/bp_loader
$ ls
bpv4_firmware_cthwre.hex  pirate-loader_lnx_bpv4
$ sudo chmod +x pirate-loader_lnx_bpv4
$ sudo chmod o+rw /dev/ttyACM0
$ ./pirate-loader_lnx_bpv4 --dev=/dev/ttyACM0 --hex=bpv4_firmware_cthwre.hex
+++++
Pirate-Loader for BP with Bootloader v4+
Loader version: 1.0.2 OS: Linux
+++++

Parsing HEX file [bpv4_firmware_cthwre.hex]
Found 22713 words (68139 bytes)
Opening serial device /dev/ttyACM0...OK
Configuring serial port settings...OK
```

- The pirate-loader\_inx\_bp4 file should output some firmware update progress information. At the end, you should see the success message, **“Firmware updated successfully :)!”**

```

Writing page 51 row 408, cc00...OK
Writing page 51 row 409, cc80...OK
Writing page 51 row 410, cd00...OK
Writing page 51 row 411, cd80...OK
Writing page 51 row 412, ce00...OK
Writing page 51 row 413, ce80...OK
Writing page 51 row 414, cf00...OK
Writing page 51 row 415, cf80...OK
Erasing page 170, 2a800...OK
Writing page 170 row 1360, 2a800...(SKIPPED by bootloader)...OK
Writing page 170 row 1361, 2a880...(SKIPPED by bootloader)...OK
Writing page 170 row 1362, 2a900...(SKIPPED by bootloader)...OK
Writing page 170 row 1363, 2a980...(SKIPPED by bootloader)...OK
Writing page 170 row 1364, 2aa00...(SKIPPED by bootloader)...OK
Writing page 170 row 1365, 2aa80...(SKIPPED by bootloader)...OK
Writing page 170 row 1366, 2ab00...(SKIPPED by bootloader)...OK
Writing page 170 row 1367, 2ab80...(SKIPPED by bootloader)...OK

Firmware updated successfully :)!
$

```

- Unplug the Bus Pirate from the PC, wait a moment and reconnect the Bus Pirate. Connect with minicom and verify that the firmware is the correct version with the ‘i’ command:

```

sudo chmod o+rw /dev/ttyACM0
minicom -b 115200 -D /dev/ttyACM0

```

```

HiZ>i
Bus Pirate v4
Community Firmware v7.1 - CTHWRE Edition [HiZ 1-WIRE UART I2C SPI 2W
DEVID:0x1019 REVID:0x0004 (24FJ256GB106 UNK)
http://dangerousprototypes.com
HiZ>

```

### 5.3 Using the FTDI Cable

- Plug in the FTDI cable and determine which device file descriptor was created using dmesg:

```
dmesg | grep -i ttyusb
```

Mine appears as /dev/ttyUSB0, but yours might have a different number. Replace “ttyUSB0” in the following steps with the proper “ttyUSBx” on your machine.

- Give permissions to /dev/ttyUSB0 and run minicom at 115200 bps:

```

sudo chmod o+rw /dev/ttyUSB0
minicom -b 115200 -D /dev/ttyUSB0

```

- Turn off hardware flow control to enable sending characters:

```

CTRL-A Z
O
“Serial port setup”
F

```

- You may be getting garbled data at this bitrate. Try using other bitrates with the -b flag when you start minicom or change the bitrate from within minicom to the proper speed.

### 5.4 Bus Pirate UART Probing (legacy)



			1-Wire	UART	I2C	SPI	JTAG	ICSP
1	MOSI	White	OWD	TX	SDA	MOSI	TDI	MOSI
2	CLK	Black		RTS	SCL	TCK	TCK	SCK
3	MISO	Brown		RX		MISO	TDO	MISO
4	CS	Red		CTS		CS	TMS	RESET
5	AUX	Orange	Auxiliary I/O, freq probe, PWM					
6	AUX1	Yellow	Auxiliary I/O1					
7	AUX2	Green	Auxiliary I/O2					
8	ADC	Blue	A/D converter, max 6V, 10b 500ksp/s					
9	Vpu	Purple	Input for pull-up resistors (0-5V)					
10	+3.3V	Gray	On-board supply, max. 150mA 150mA					
11	+5V	White	On-board supply, max. 150mA 150mA					
12	GND	Black	Ground to test circuit					

**Note:** There are two dominating hardware versions of the Bus Pirate: BPv3.6 and BPv4.0. They differ in some key ways: Number of pins (10 vs. 12), pin assignments, microprocessor, and external EEPROM in the BPv4. Make sure you are using the right pin-out diagram for your BP!

2. Install minicom if needed:

```
$> sudo apt install minicom
```

3. Plug in the Bus Pirate USB and determine which device file descriptor was created in /dev/ using dmesg:

```
$> sudo dmesg | tail | grep tty
```

```
$ sudo dmesg | tail | grep tty
[ 1494.110513] cdc_acm 1-2:1.0: ttyACM0: USB ACM device
$
```

**Note:** Mine appears as /dev/ttyACM0, but yours might be different with a different name (e.g. /dev/ttyUSB0) or number (e.g. /dev/ttyACM1). Replace “ttyACM0” in the following steps with the proper “ttyXXXX” on your machine.

4. Give permissions to /dev/ttyACM0 and run minicom at 115200 bps:

```
$> sudo chmod o+rw /dev/ttyACM0
```

```
$> minicom -b 115200 -D /dev/ttyACM0
```

```
$ sudo chmod o+rw /dev/ttyACM0
$ minicom -b 115200 -D /dev/ttyACM0
```

```
Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on May 6 2018, 08:02:47.
Port /dev/ttyACM0, 16:51:28

Press CTRL-A Z for help on special keys

HiZ>
HiZ>
HiZ>
```

**Note:** Hit <Enter> a few times, you should be greeted with a Bus Pirate prompt: “HiZ>”

5. Type ‘i’ and hit <Enter> to display the version info and make sure the Bus Pirate firmware is updated to the “Community Firmware v7.1 - CTHWRE Edition”

```
HiZ>i
Bus Pirate v4
Community Firmware v7.1 - CTHWRE Edition [HiZ 1-WIRE
DEVID:0x1019 REVID:0x0004 (24FJ256GB106 UNK)
http://dangerousprototypes.com
HiZ>
```

**Note:** If your Bus Pirate does not have the proper version, go to Section 5 Extras > Upgrade Bus Pirate Firmware and follow the steps there before continuing.

### 5.4.1 Using Bus Pirate for UART Probing

1. Put the Bus Pirate into UART mode and configure it to monitor or bridge the target's UART interface. There's a good overview of this mode on the official site <http://dangerousprototypes.com/docs/UART>

- a. Set UART mode

```
HiZ> m
```

```
(1)> 3
```

```
HiZ>m
1. HiZ
2. 1-WIRE
3. UART
4. I2C
5. SPI
6. 2WIRE
7. 3WIRE
8. KEYB
9. PIC
10. DIO
x. exit(without change)

(1)>3
Set serial port speed: (bps)
1. 300
2. 1200
3. 2400
4. 4800
5. 9600
6. 19200
7. 38400
8. 57600
9. 115200
10. Input Custom BAUD
11. Auto-Baud Detection (Activity Required)

(1)>
```

- b. Set baud rate, parity bits, stop bits, and error bits. Use the measured values from earlier, or make a guess. The auto-baud detection mode doesn't work very well. When prompted, select "**Normal (H=3.3V, L=GND)**" for the output type.
- c. Start the power supplies with command 'W' then enter "live monitor" mode with "(2)"

```
UART> W
```

```
UART> (2)
```

```
UART>W
POWER SUPPLIES ON
Clutch engaged!!!
UART>(2)
Raw UART input
Any key to exit
```

- d. To view the available modes, use "(0)"

```
UART> (0)
```

```

UART>(0)
0.Macro menu
1.Transparent bridge
2.Live monitor
3.Bridge with flow control
4.Auto Baud Detection (Activity Needed)

```

- e. To interact with the board, use “transparent bridge” mode with “(1)”. Make sure the power supplies are on with ‘W’ before entering into this mode!

```

UART> W
UART> (1)

```

```

y
UART>W
POWER SUPPLIES ON
Clutch engaged!!!
UART>(1)
UART bridge
Normal to exit
Are you sure? y

```

#### Useful Minicom commands:

All commands from the main Minicom window can be accessed from the help window which can be accessed by first pressing CTRL-A followed by Z. Some have shortcuts that begin with CTRL-A followed by a single key press.

Minicom Command	Help Window Sequence	Shortcut
Help	CTRL-A, Z	
Exit	CTRL-A, Z, X	CTRL-A, X
Configure comm. Parameters (Baudrate, data bits, stop bits, parity)	CTRL-A, Z, O	CTRL-A, P
Toggle local “echo” on/off	CTRL-A, Z, E	CTRL-A, E
Clear screen	CTRL-A, Z, C	CTRL-A, C

#### Useful Bus Pirate commands:

Bus Pirate Command	Key
Help	?
Reset	#
Change mode	m
Show version info	i
Jump to bootloader	\$

## 5.5 Bus Pirate with AVRdude for ICSP Memory Read/Write

The microcontroller on the Smart Sensor Simulator (SSS) Daughter Board (DB) uses an AVR instruction set architecture (ISA). Therefore, we need to look for an ICSP port to begin on-chip debugging. ICSP stands for In-Circuit Serial Programming. It is a proprietary standard for on-chip debugging similar to JTAG ([https://en.wikipedia.org/wiki/In-system\\_programming](https://en.wikipedia.org/wiki/In-system_programming)). Additionally, the open-source ecosystem has mature support for the AVR architecture. Because of this, we can use extremely inexpensive or free tools to attach to the SSS DB’s On-Chip Debugging port.

### 5.5.1 Class Goals



1. Dump device firmware

Target Info Name	Target Info Value
Size of firmware	
16 bytes of FLASH starting at address 0x0	
16 bytes of EEPROM starting at address 0x0	

2. Modify firmware and write it back

### 5.5.2 Setup AVRdude & Bus Pirate

We will be using the Bus Pirate v4.0 as our ICSP interface device and AVRdude as our interface software. AVRdude, once setup, will use the Bus Pirate as a protocol bridge to the AVR core running inside the Atmel ATmega 2560.

1. Download, compile, and install AVRdude.
  - a. Clone the latest avrdude build scripts from the arduino git repo:

```
$> git clone https://github.com/arduino/avrdude-build-script.git
```

- b. Install dependencies

```
$> sudo apt install build-essential libtool automake pkg-config subversion zip flex bison
gperf libelf-dev libusb-dev libftdi-dev xsftproc
```

- c. Move into the repository directory and run the script to kickoff the build:

```
$> cd avrdude-build-script
$> ./avrdude-6.3.build.bash
```

**Note:** The patch step detects some issues, force the patch by pressing 'y' when prompted with "Reversed (or previously applied) patch detected! Assume -R? [n]"

- d. Move to where avrdude was built and make sure it was successful

```
$> cd avrdude-6.3
$> sudo chmod +x avrdude
$> ./avrdude -v
```

```
$ cd avrdude-6.3
$ sudo chmod +x avrdude
$ ./avrdude -v

avrdude: Version 6.3-20190610
Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/
Copyright (c) 2007-2014 Joerg Wunsch
```

- e. Copy the binary to your user binary directory so you can call it from anywhere in bash.

```
$> sudo cp avrdude /usr/local/bin/.
```

2. Wire the Bus Pirate to the ICSP pins of the Atmel microcontroller.

### 5.5.3 Using Bus Pirate with AVRdude for ICSP Memory Read/Write

5. Use the Bus Pirate as the programmer with avrdude in terminal mode



```
$> sudo chmod o+rw /dev/ttyACM0
$> avrdude -p m2560 -P /dev/ttyACM0 -c buspirate -t
$ avrdude -p m2560 -P /dev/ttyACM0 -c buspirate -t

Attempting to initiate BusPirate binary mode...
avrdude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.03s

avrdude: Device signature = 0x1e9801 (probably m2560)
avrdude>
```

**Note:** If this doesn't work the first time, hit Ctrl+C to cancel, then try it again. If that doesn't work, try using Bus Pirate in bitbang mode by substituting "buspirate" with "buspirate\_bb"

**Note:** You can easily use avrdude with a microcontroller specific on-chip debugger like the avrispmkii by changing the "-c" flag to "avrispmkII" and dropping the "-P" flag.

6. Read an address in flash memory from avrdude terminal mode

```
avrdude> dump flash 0x5040 16
avrdude> dump flash 0x5040 16
>>> dump flash 0x5040 16
5040 53 6d 61 72 74 20 53 65 6e 73 6f 72 20 53 69 6d |Smart Sensor Sim|
```

7. You could read all flash this way, but it's easier to do it in one shot from the command line. Exit terminal mode then read all flash firmware into a file

```
avrdude> quit
$> avrdude -p m2560 -P /dev/ttyACM0 -c buspirate -x serial_recv_timeout=10 -x spifreq=3 -U
flash:r:sssdb_flash.bin:r
$ avrdude -p m2560 -P /dev/ttyACM0 -c buspirate -x serial_recv_timeout=10 -x spi
freq=3 -U flash:r:sssdb_flash.bin:r

Attempting to initiate BusPirate binary mode...
avrdude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.03s

avrdude: Device signature = 0x1e9801 (probably m2560)
avrdude: reading flash memory:

Reading | ##### | 51% 9.22s
```

8. Let's change part of the firmware and upload it back. Any strings that appear in the UART boot console are good candidates because we can ensure our changes really did take affect.

```
$> vim sssdb_flash.bin
$> avrdude -p m560 -P /dev/ttyACM0 -c buspirate -x serial_recv_timeout=10 -x spifreq=3 -U
flash:w:sssdb_flash.bin
```

```

$ vim sssdb_flash.bin
$ avrdude -p m2560 -P /dev/ttyACM0 -c buspirate -x serial_recv_timeout=10 -x spi
freq=3 -U flash:w:sssdb_flash.bin

avrdude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.03s

avrdude: Device signature = 0x1e9801 (probably m2560)
avrdude: NOTE: "flash" memory has been specified, an erase cycle will be perform
ed
        To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "sssdb_flash.bin"
avrdude: input file sssdb_flash.bin auto detected as raw binary
avrdude: writing flash (261408 bytes):

Writing | ##### | 100% 48.59s

avrdude: 261408 bytes of flash written
avrdude: verifying flash memory against sssdb_flash.bin:
avrdude: load data flash data from input file sssdb_flash.bin:
avrdude: input file sssdb_flash.bin auto detected as raw binary
avrdude: input file sssdb_flash.bin contains 261408 bytes
avrdude: reading on-chip flash data:

Reading | ##### | 100% 115.14s

avrdude: verifying ...
avrdude: 261408 bytes of flash verified

avrdude: safemode: Fuses OK (E:FD, H:D8, L:BF)
avrdude done. Thank you.

```

#### Useful AVRdude commands:

AVRdude Command	Command
Help	?
Microcontroller info	part
Read memory (e.g. fuses, eeprom, flash)	dump
Write memory (e.g. fuses, eeprom, flash)	write

## 5.6 JTAG with a Bus Pirate

**Note:** The instructions for accessing JTAG with a Bus Pirate can be found in the CyberTruck Challenge 2017 Hardware Reverse Engineering Student Workbook and won't be covered during this year's class. I wanted to keep the information here for those curious who might want to use JTAG during the event.

Joint Test Action Group (JTAG) is a standard for “boundary scan” testing and in-circuit debugging, among other things. JTAG uses Serial Peripheral Interface (SPI, pronounced “spy”) for the actual data transmission between the members of the scan chain. Because the Bus Pirate speaks SPI, it can also speak JTAG. Similar to ICSP above, JTAG will allow us to dump the processor's memory and perform interactive control of the running program.

Interactive debugging can be more useful than static software reverse engineering. During static SW RE of a binary, a reverse engineer must attempt to follow the control flow, but during interactive debugging, the processor follows the control flow itself and the reverse engineer can hitch along for the ride. This ability enables a SW RE engineer to find interesting areas of code such as proprietary crypto algorithms, the addresses of keys, and perhaps even ephemeral

security data (e.g. IoT device that only stores crypto material in RAM) even when obfuscated. This is done by setting breakpoints in interesting blocks of code then providing the proper stimulus to cause the hardware to execute the interesting security function. Once the breakpoint is hit, the debugger can read or write memory and registers at will.

The following tools can be used to interface with a JTAG port:

- **Bus Pirate** – The Bus Pirate is the hardware interface that attaches to the JTAG port on one side and the host PC via USB on the other.
- **OpenOCD** (<https://sourceforge.net/p/openocd/code/ci/master/tree/>) – OpenOCD is an open-source software tool that supports a range of hardware interfaces for JTAG as well as a wide-variety of targets.
- **GDB** (<https://www.gnu.org/software/gdb/>) – The GNU Debugger is a free software command line software debugger.
- **Eclipse** (<https://www.eclipse.org/downloads/>) – Eclipse is an open-source IDE/debugger framework, which provides a visual interface for software debugging.

## 5.7 Glossary

- **Breakpoint** – A **breakpoint** is a hardware (or software) trigger to stop the processor when performing on-chip debugging.
- **Baud Rate** - The **baud rate** of a digital bus defines the number of times a signal can change on a transmission line per second. This is different from bit rate where bit rate measures the number of logical '0's or '1's that can be transmitted per second. Most digital communication uses two signal states (low/high) with each representing a single bit (0/1), so in those cases baud rate and bit rate are equivalent. Measured in symbols per second.
- **Bit Rate** - The **bit rate** of a digital bus defines the number of times a logical '0' or '1' can be sent on a transmission line per second. This is different from baud rate where baud rate measures the number of symbols that can be transmitted per second. Most digital communication uses two signal states (low/high) with each representing a single bit (0/1), so in those cases baud rate and bit rate are equivalent. Measured in bits per second.
- **Device file** – A **device file** is a special file in Unix and Linux which represents a device connection rather than a file in the filesystem on a disk.
- **GCC** – The **GNU Compiler Collection (GCC)** is a free and open-source compiler that is part of the GNU project.
- **GDB** – The **GNU Debugger** is a free and open-source command-line software debugger that is part of the GNU project.
- **ICD/OCD** – In-circuit debugging (ICD) and on-chip debugging (OCD) are terms used for debugging hardware in-situ. The debugging circuitry is installed in the hardware inside of or alongside the microcontroller/processor.
- **JTAG** – **JTAG** stands for Joint Test Action Group. JTAG is a standard for simple hardware testing and in-situ (ICD/OCD) debugging of hardware.
- **Microcontroller** – A **microcontroller** is a (usually small and slow) system-on-a-chip (SoC) packaged as a single integrated circuit (IC) which includes one or more processor cores, read-only memory (ROM), random-access memory (RAM) and peripherals, such as those for serial communications or analog/digital conversion.
- **Multimeter** – A **multimeter** is an electronics engineering tool used for measuring simple properties like voltage and current. Unlike an oscilloscope, a multimeter generally cannot analyze time-varying signals.
- **Oscilloscope** – An **oscilloscope** is an electronics engineering tool for analyzing analog signals in real-time.
- **PCB** – A **Printed Circuit Board (PCB)** is a computer board with various electrical interconnects (called traces) laid into a single- or multi-layer substrate to form circuits connecting the discrete components, such as integrated circuits (ICs) and passives like resistors and capacitors.
- **PHY** – Short for **physical** interface that implements OSI layer 1.

- **Silk screen** – **Silk screen** is a nonfunctional annotation layer on the top and/or bottom of a PCB which often shows component identifiers and sometimes even comments.
- **TTL** – **Transistor-transistor logic** is a common PHY for UART serial communications.
- **Tuning** – In automotive electronics, **tuning** is the process of performing hardware and/or software modifications to the vehicle's control systems (such as an engine controller) to achieve a different performance profile than the vehicle was designed and shipped with.
- **UART** – A **Universal Asynchronous Receiver/Transmitter (UART)** is a computer peripheral technology for serial data communications, commonly employed to create a serial terminal for debugging and development.