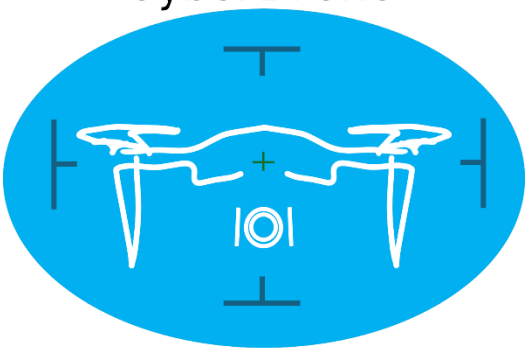
CyberDrone



Challenge

Hardware Reverse Engineering

2024 Student Workbook

September 24, 2024

Instructor: Bill Hass
<billhass@umich.edu>

Source: https://github.com/bhass1/cthwre

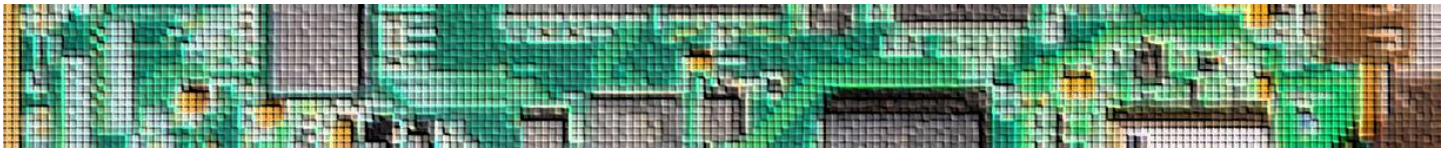


Table of Contents

1 Welcome (~10 minutes)

1.1 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 (HW) 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 diagram; (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 diagram

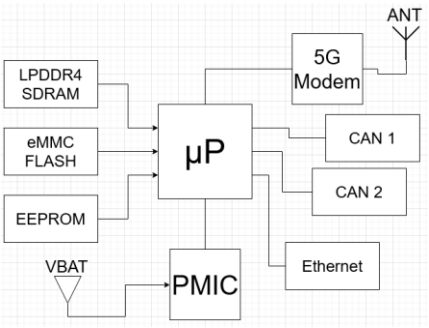


Figure 1: Example functional block diagram

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

Figure 2: Common electronics distributors

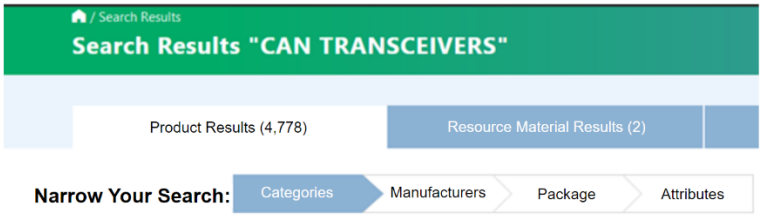


Figure 3: Seach for CAN transceiver from electronics distributor website

(3) Circuit design, layout, and routing

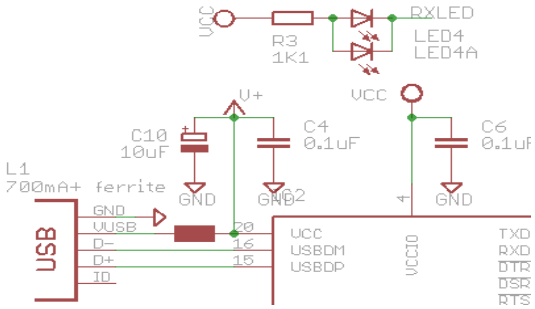


Figure 4: Example circuit schematic

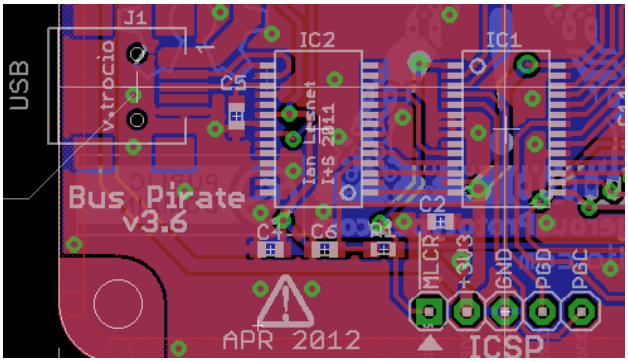


Figure 5: Example PCB assembly drawing

(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.

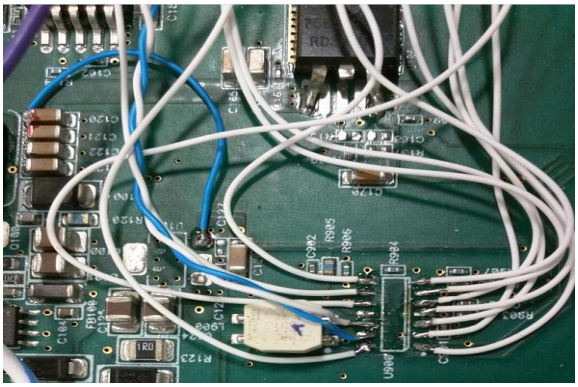


Figure 6: Example board rework

(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
- ***Extract Device Software:*** Bootloader, OS, configs, filesystems, binaries, libraries, etc.

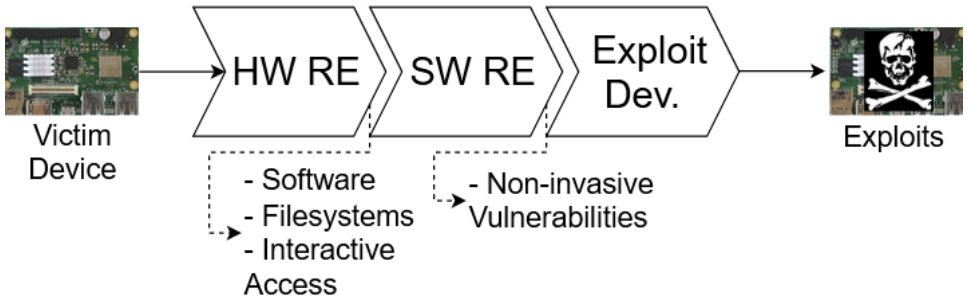


Figure 7: A common exploit development workflow: HW RE yields firmware; SW RE yields a vulnerability; and a vulnerability yields an exploit.

This HW RE course is geared towards assisting the software reverse engineering (SW RE) process. SW RE is useful for finding vulnerabilities or weaknesses in a software implementation which can lead to fixes or non-invasive exploits via exploit development. But the SW RE engineer needs access to the software first, so HW RE techniques can be used to obtain software from a target device (*It's usually better to search online for a software download first!*).

What if I just want to hack my own hardware?

Hunting for exploits to PWN someone else's device is not the only reason to reverse engineer a piece of hardware. HW:RE can be needed to fix a broken device, and local attacks alone can gain additional or altered functionality. Owners of video game consoles modify the hardware and software of their consoles to allow for homebrewed games (and enable piracy); Owners of vehicles use hardware and software modifications to block GPS & telematics tracking, "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.

Bottom Line / Inspiration: Physical attackers always win

1.2 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)

1.2.1 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.

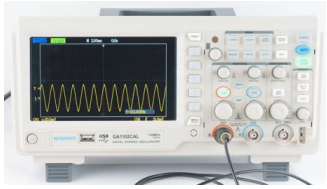
1.2.2 Soldering station

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.

1.2.3 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^2C, 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.

1.2.4 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.

1.2.5 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.

1.2.6 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.

1.2.7 Microcontroller-specific Debugger/Programmer

The Tigard 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.

2 First Contact (~40 minutes)

First contact with a new component often involves carefully observing how it is constructed by visual inspection, poking, prying, and prodding. Like peeling back a metal onion, you may need to unscrew fasteners, peel apart adhesives, and pry apart clips to get to the electronics within. **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 2.2 Visual Survey)
2) Identify at least two communication interfaces and how to connect to them (Section 2.2 Visual Survey)
3) Create a BOM & assembly drawing (Section 2.3 Chip Identification)
4) Identify microprocessors/microcontrollers (Section 2.3 Chip Identification)
5) Identify power hookup and power domains (Section 2.5 Passive Probing)
6) Locate microprocessor/microcontroller programming interface (Section 2.5 Passive Probing)

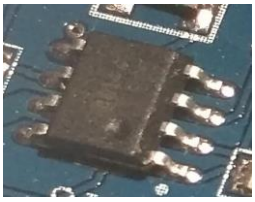
2.2 Visual Survey

As you gain access to the electronics, take some time to look carefully at them to see if you recognize anything or see any patterns. If you look closely, you will see a white printing of letters and numbers on the green surface of the PCB called a silkscreen. Like comments in software code, hardware engineers can use silkscreen to comment on the PCB. Commonly, silkscreen is used during development and debugging for adding reference designators so engineers can easily reference a board's components (e.g. R30 needs to be 125kΩ instead of 250kΩ). An "R" might stand for "resistor," "C" for "capacitor," "L" for "inductor," "J" for "jack," "TP" for "test point," and so-on. IEEE 315 standard has a list of these reference designators. Even without silkscreen, you can learn a lot about a system just by looking at it.

- How are components mounted?

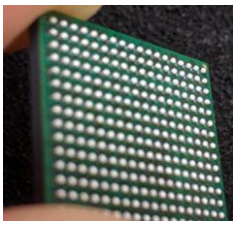
  Through-hole is your friend. Flat pack components expose all pins. BGA is your enemy.

Through-Hole



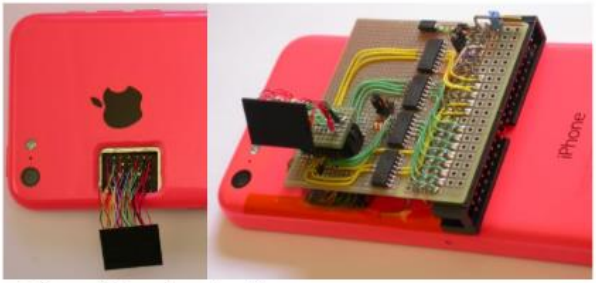
SOIC

QFN



BGA

Figure 8: Example component packaging



© 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.

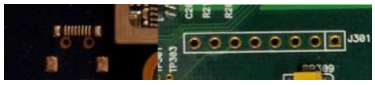


Figure 10: Interesting areas of a target device

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

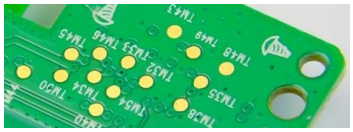


Figure 11: Groups of test points on a PCB

- 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.

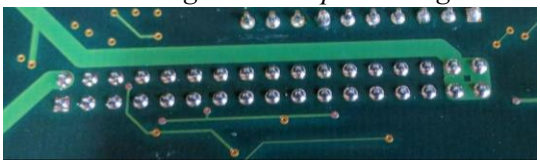


Figure 12: Fat versus skinny traces on a PCB

- 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.

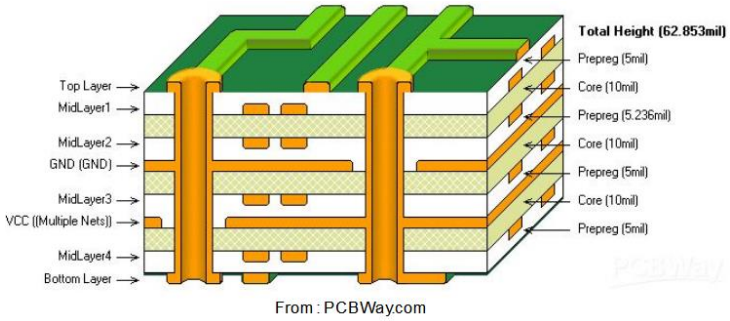


Figure 13: An 8-layer PCB

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

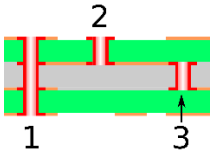


Figure 14: Types of vias; (1) via; (2) blind-via; (3) hidden-via

2.3 Chip Identification

An example of a partial "Bill-of-Materials" (BOM) is shown in Figure 15. This is one of the HW:RE's first major contributions to the overall security assessment, and it's valuable as both a lookup table for the physical components and a pointer to the technical details.

Silkscreen/ID	IC Marking	Vendor	Component Type	Datasheet
U601	TIM-5H-0-004	Ublox	GPS Transceiver	https://u-blox.com/c
U805	POWR1014	Lattice	Power Supply Supervisor	lattice_latt-s-a000138

Figure 15: Example BOM sufficient for HW:RE

It is also a good time to start an assembly drawing / block diagram since you will be systematically noting down components during this time. For the assembly drawing / block diagram, I like to take a picture of the target and draw over it in PowerPoint or GIMP. An example is shown in Figure 16.

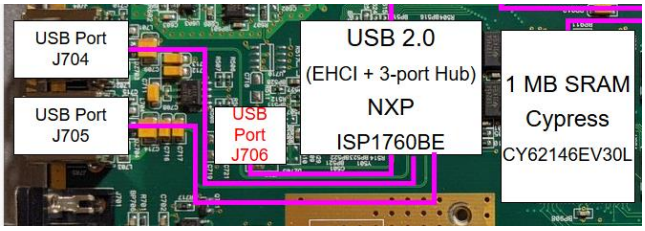


Figure 16: Example assembly drawing / block diagram with logical blocks aligned to phyusical components.

- 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.*

2.3.1 Step-by-Step Advice

1. Bigger chips on the circuit board are usually more complex and interesting things like processors and memory, so it can be useful to start with those.
2. Write down the silkscreen reference or ID and the IC marking.
3. Enter the IC marking directly into a search engine. Useful resources: https://duckduckgo.com/; https://octopart.com; http://www.smdmark.com/en-US/; https://www.elnec.com/en/support/ic-logos/.
4. Try to find the datasheet on the internet and use it to fill in the other columns.
5. Repeat until you identify every chip on the board. For this exercise it's sufficient to find the flash, UART transceiver, and the MCU.

2.4 Parsing Datasheets

Datasheets contain a plethora of technical information, but not all of it is relevant for HW:RE right away. Keyword searches are crucial for navigating a datasheet. Use the datasheets to accomplish the goals 2-4 of listed in Section 2.1.2.

2.4.1 Step-by-Step Advice

1. Search for a block diagram which will shed some light on what is within the IC and how it may be connected to other components on the PCB (e.g. via I2C, SPI, UART, or USB).
2. Do a keyword search to see if anything interesting turns up using: debug, program, jtag, and security.
3. Look for pin descriptions and how they map to the physical package. For instance, this will tell you where a UART RX/TX or JTAG TDI, TDO, TCK interface is located physically on the device.
4. Look at recommended operating conditions so you can know how to safely power the device later on.

2.5 Passive Probing

Always verify your assumptions! **With the board unpowered still**, we will use a multimeter to confirm what we learned in the previous steps and further reverse engineer the circuitry to better understand the board function. You should also draw schematics by hand like the one in Figure 18 during this phase as you develop an understanding of the circuits.

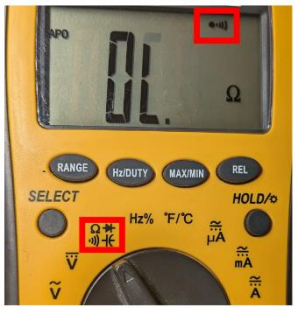


Figure 17: Close cropped view of multimeter configured for continuity testing

Put the multimeter in "continuity test mode." You can find this mode on your multimeter by looking for the "Wi-Fi" symbol. See Figure 17 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 19 shows a basic circuit and expected continuity results.

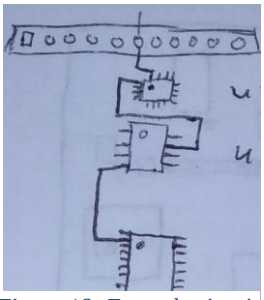


Figure 18: Example circuit drawing

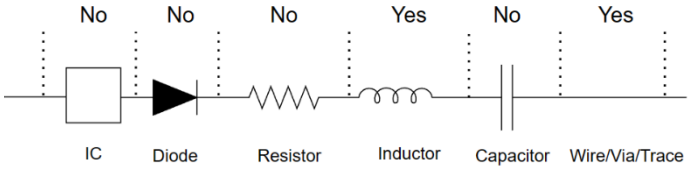
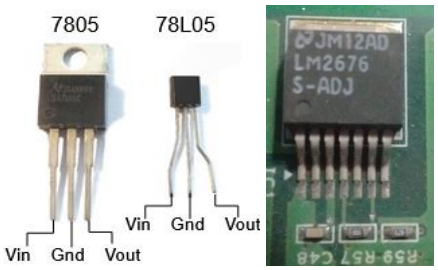


Figure 19: Example circuit showing continuity test results

- Where does the board get its power, and how is it routed and converted?
  Identify voltage regulator connections, and check where big traces go.



Examples of voltage regulators

- 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.

2.6 Section Checkpoint

2.6.1 What package is the main microcontroller?	
2.6.2 What is the silkscreen code of the regulator powering the main microcontroller? e.g. “Uxxx”	
2.6.3 Where will you connect 12V power?	
2.6.4 What is the BOM entry (Silkscreen/ID, IC Marking, Vendor, Component Type, Datasheet) for the non-volatile memory component(s)	
2.6.5 What is the BOM entry (Silkscreen/ID, IC Marking, Vendor, Component Type, Datasheet) for the UART transceiver?	
2.6.6 What pins (e.g. pin 1 & pin 3) are mapped to the UART transceiver’s TXA/RXA?	
2.6.7 What is the BOM entry (Silkscreen/ID, IC Marking, Vendor, Component Type, Datasheet) for the microcontroller?	
2.6.8 What pins (e.g. pin 1, pin 2, & pin 3) are mapped to the microcontroller’s JTAG TDI, TDO, TCK?	

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^{O}C / ~480^{O}F); 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.

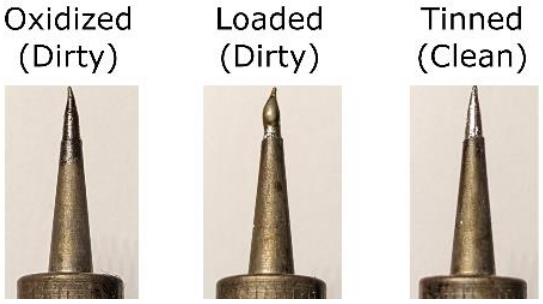


Figure 20: Three stages of cleaning a soldering iron tip.

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

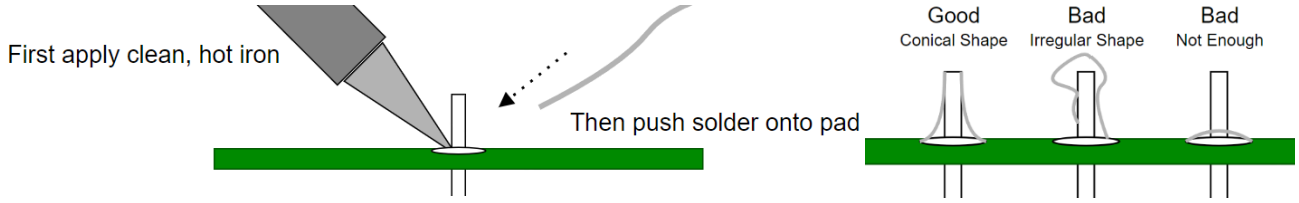


Figure 21: Basic steps of soldering through-hole components & quality of solder joints.

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

Always verify your assumptions! 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!

- 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).
- 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.

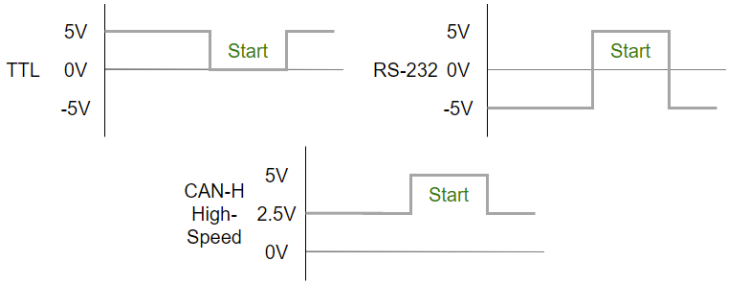


Figure 22: Steady-state and start bits of common digital communication protocols

4.2.1 Multimeter Probing Checkpoint Questions

4.2.1.1. What is the measured output voltage of the U803 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. For this class, we will focus on using the logic analyzer.

- Find digital network buses.
  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.
  *For asynchronous digital networks (e.g. CAN or UART) a baud rate is needed. 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. $(8.68us)^{-1}$ = ~115200bps). Section 4.4 UART Probing will walk you through this.*

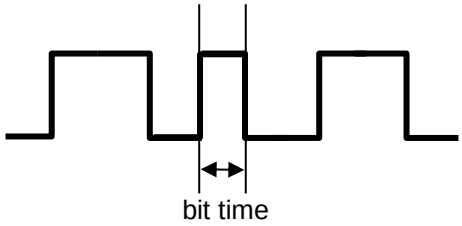
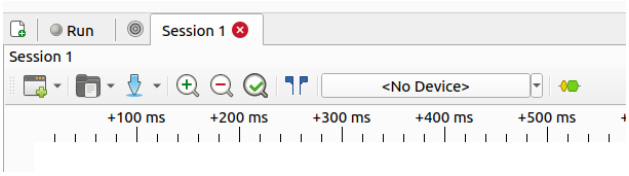


Figure 23: Measuring bit time to calculate baud rate

- Decode signals
  Logic analyzers and most oscilloscopes are able to decode signals in real-time. Section 4.4 UART Probing will walk you through this.

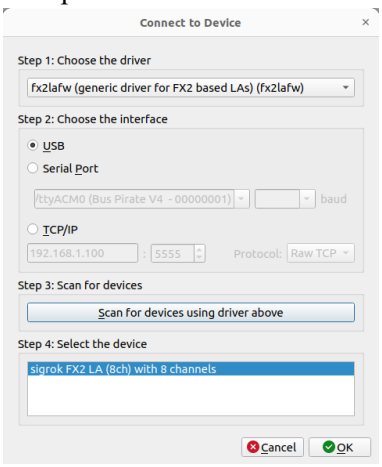
4.3.1 Setup the BitMagic Basic Logic Analyzer

1. 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).
    1. 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.
2. Open the "PulseView" GUI application either from the app drawer or by typing `pulseview` in the terminal. You should see a similar screen.



3. Connect to Device…

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



5. Now, you are ready to collect some data in the next section.

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 Goal

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.

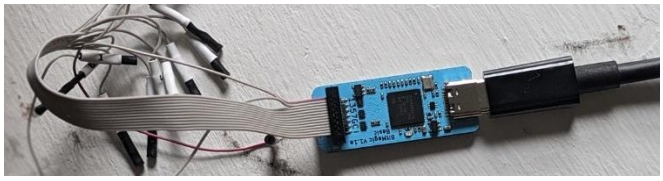


Figure 24: Close cropped view of BitMagic Basic with flywire pigtail

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)

1. 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.
   a. **<u>Note:</u>** *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* ☺

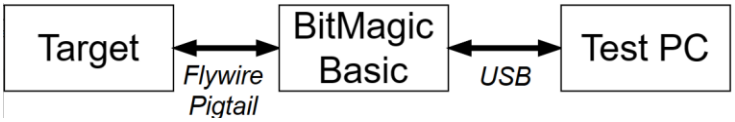
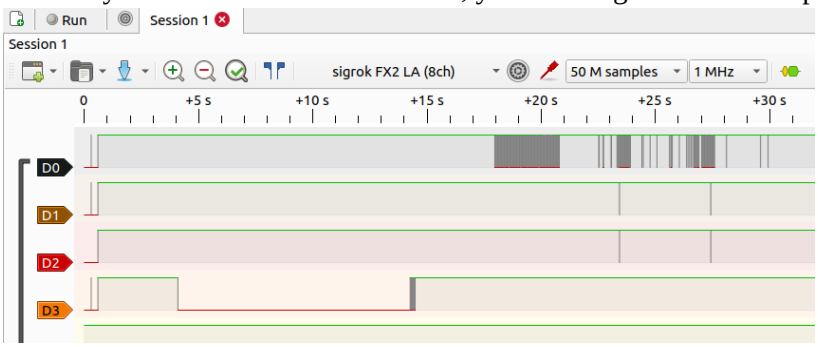


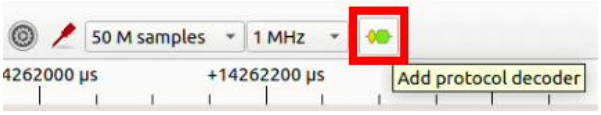
Figure 26: Block diagram for UART Rx using BitMagic Basic Logic Analyzer

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

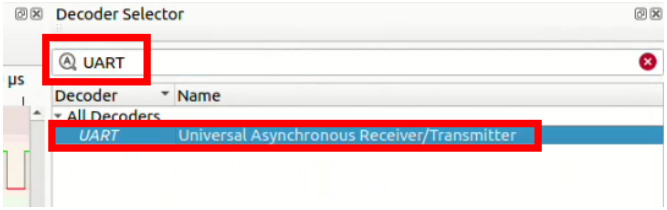


2. Add a decoder for UART for a channel of your choice. You can add more than one.

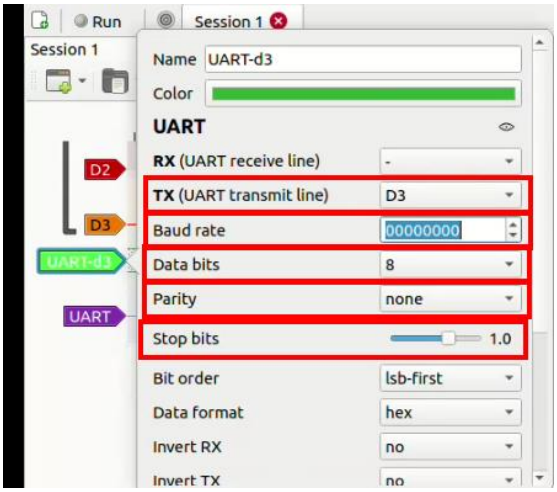
a. Find the toolbar icon for “Add protocol decoder” and click it.



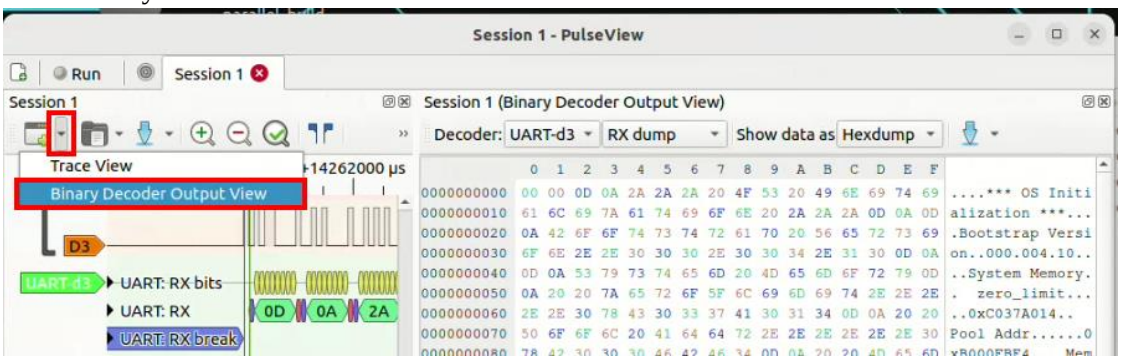
b. In the new panel, there are many options available. Filter and select UART.



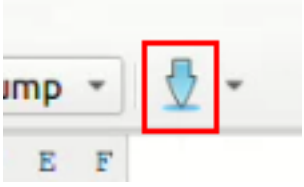
c. Configure the UART parameters for the UART protocol decoder by left-clicking then filling the information. The most common configuration fields are outlined in red below. Use what you learned in Section 4.3 for configuring the baudrate.



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

0.2. What is the Mobile Equipment Identifier (MEID) of your target?	
0.3. 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”). 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.

If you have a Tigard on hand, you can use that instead! While it doesn’t support RS232 or RS485, it does support different voltage levels for TTL UART, and several other important protocols. We will be using the Tigard and BitMagic Basic for this section setup as shown in Figure 27 below.

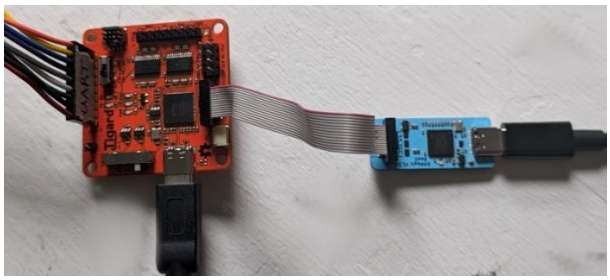


Figure 27: Close cropped view of Tigard with UART flywires connected to BitMagic Basic

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 28 for a block diagram.

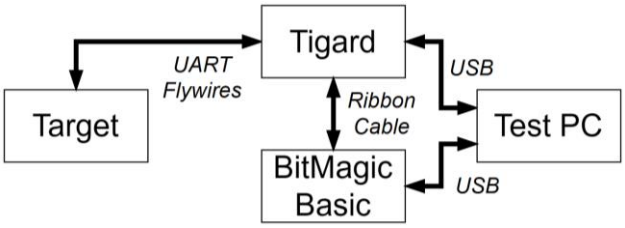


Figure 28: Block diagram for UART Tx using Tigard with BitMagic Basic

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.

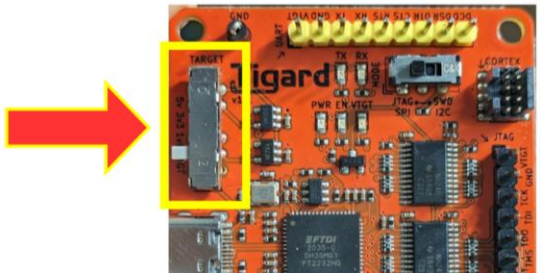


Figure 29: Close cropped view of Tigard's TARGET voltage switch

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 the 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
  - Extract firmware, modify it, write it back
  - 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 30: 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 27 above. This time, the Tigard will be our JTAG interface device and OpenOCD will be our interface software.Once setup, will use the Tigard as a protocol bridge to the ARM922T core running inside the NXP LH7A400.

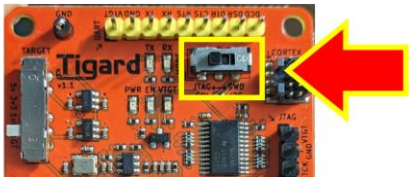
1. Check that OpenOCD is installed with `openocd -v`

```
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 31 for a block diagram.

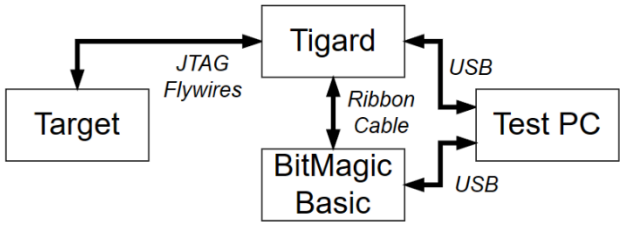


Figure 31: Block diagram for JTAG using Tigard with BitMagic Basic

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
```

3. 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.

4. 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.

1. 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`
2. Our flash won't respond to "CFI" driver commands unless it has been reset, so issue a `reset init`
3. 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`
4. Some flash banks have write protection. Check this with `flash info 0`
    a. 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.
5. Now write the modified flash binary to flash! `flash write_bank 0 test_dump_lh7a400.bin`