CyberTruck Challenge

Hardware Reverse Engineering

Student Workbook

Russ Bielawski, Bill Hass

6/26/2017

1. Introduction

Our hardware reverse engineering course is geared towards assisting the software reverse engineering process. Software reverse engineering is useful for would-be attackers and security researchers (we'll just use the phrase "reverse engineers" from now on) to hunt for vulnerabilities. To reverse engineer software for a device, however, the reverse engineer needs access to the software. A reverse engineer can use hardware reverse engineering techniques to obtain that software from a device in their possession. 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 attempt to find vulnerabilities in a device's implementation that may allow for remote exploitation on other devices of a same type.

Hunting for remote vulnerabilities is not necessarily the only reason to reverse engineer a piece of hardware (or, part of a piece of a hardware). Local attacks alone are often attractive to owners seeking to achieve additional or altered functionality from a device they own. Owners of cellular phones reprogram the firmware to access new features (or remove unwanted features), owners of video game consoles modify the hardware and software of their consoles to allow for homebrewed games (and enable piracy), and owners of automobiles use hardware and software modifications to "tune" their vehicles to achieve better performance than they were shipped with.

Figure 1: For devices that a reverse engineer has physical access to, including the vast majority of embedded systems, a reverse engineer can use hardware reverse engineering to obtain software (in binary form) for reverse engineering. Software reverse engineering can be used to find vulnerabilities that may enable further exploitation, possibly remote exploitation. This is a powerful ability, because, in the usual case, a remotely-exploitable vulnerability in an embedded system will apply to all similar devices in the field.

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

- Basic use of electronics tools
  - Multimeter
  - Soldering iron
  - Oscilloscope
  - Logic analyzer
  - Serial terminal decoder

    - On-chip debugger
- Circuit identification – The process of reconstructing knowledge about the circuitry on the printed-circuited board (PCB) of the victim hardware.
- Board modification – The process of modifying the PCB to enable breakout of interesting signals or change functionality to assist in hardware reverse engineering. (For this class, this means light soldering.)
- Serial data interfacing – The process of identifying interesting serial data interconnects on the victim hardware and tapping into them for inspection, analysis and, possibly, injection of commands or data.
- In-circuit debugging – The process of attaching to the built-in debugging circuitry in the board and/or processor(s) and accessing memory and controlling the processor in real-time.

1.1 Tools Introduction

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
- Modifying hardware to breakout signals for easy access or to create new connections and circuits
- Analysis of analog signals (sine waves or other signals with more than two basic voltage states)
- Analysis of digital signals (signals which should have two basic voltage states)
- Decoding of encoded data transmitted on digital signals
- Injection of commands and/or data into digital communications links on the hardware
- Real-time analysis and control of the processors (or FPGAs, etc.) on the hardware

1.1.1 Multimeter

A multimeter is a tool which can measure voltage and current. In addition, multimeters can perfom what is called "continuity testing," or probing to see if two points are electrically connected to one another. This functionality is particularly useful for tracing an unknown circuit on a board without documentation.

Most decent multimeters will have an auto-range function, so you will not need to think about the order of magnitude of the signal you intend to measure. However, if your multimeter does not support auto-ranging, you will need to take a rough estimate before setting its mode.

1.1.2 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.

1.1.3 Logic analyzer

A logic analyzer is like an oscilloscope, but it can only monitor digital signals. Logic analyzers are generally cheaper than oscilloscopes and usually support more channels. Most logic analyzers do not perform real-time monitoring, another difference from oscilloscopes. Rather, the only mode of operation is to set a trigger and look at what was captured after the trigger fired.

1.1.4 Soldering station

A soldering station is useful for modifying (and building) hardware. New connections can be added so that oscilloscope or logic analyzer probes maybe attached more easily or unpopulated components added for reverse engineering. A soldering station is essentially necessary for anything but the most trivial hardware reverse engineering tasks.

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

A USB-to-serial adapter allows for decoding of serial links (UART links) on a PC as well as injecting commands. This is more useful than a logic analyzer alone, because it allows the reverse engineer to interact with the serial port directly in text form. These devices are sometimes called “FTDI cables,” because the company FTDI has a corner on the market of USB-to-serial adapter integrated circuits.

1.1.6 Bus Pirate

The Bus Pirate is a neat tool which can be used to perform serial data decoding and injection for several protocols including Universal Asynchronous Receiver/Transmitter (UART), Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I^2C). Because JTAG, an on-chip debugging standard, is SPI-based, the Bus Pirate can also interface with JTAG connections to perform debugging functionality for microcontrollers and other devices with JTAG support.

1.1.7 Miscellaneous Parts

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

- Screwdrivers
- Razor blades
- Tweezers
- Pliers
- Strippers
- Q-tips
- Paper clips
- Soldering iron
- Multimeter
- DC power supply
  - Battery
  - Wall wart
  - Bench-top
- Jumper wires
- Headers
- Patch wire
- Micro clamp probes
- Solder
- Desoldering wick or pump
- Flux
- Linux computer
- USB A, B, mini, micro
- UART-to-USB (FTDI) cable
- Bus Pirate or JTAGGER

2. Circuit Identification (~30 minutes)

Before even powering up the hardware, look at the hardware itself. Circuit identification involves carefully taking apart the hardware or parts of the hardware to get to the circuitry and performing circuitry reconnaissance to identify areas of interest.

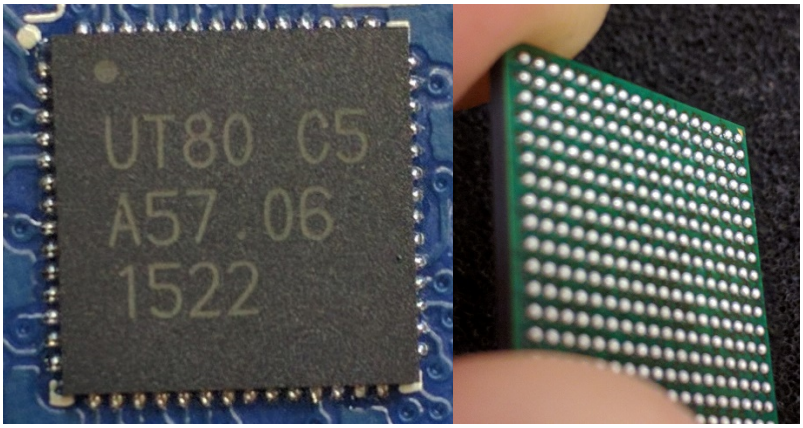
2.1 Visual Analysis

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. Therefore, it is useful to look at the board to begin to get an idea of what does what. Identify the major components on the PeopleNet G3 board and try to find their manuals on the internet.

Initial lay-of-the-land inspection.

- How are components mounted?

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



- Are there barriers or protections in place?

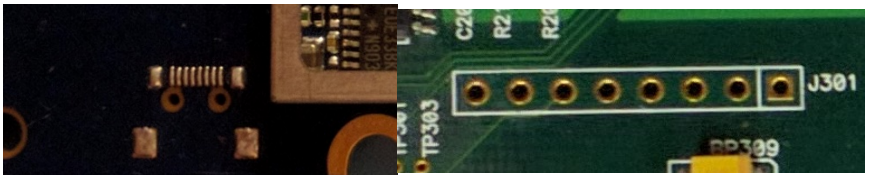
  EMF shielding and robustness coatings can make our job difficult.

- What are the populated interfaces?

  Things like USB, vehicle connectors, and hidden connectors.

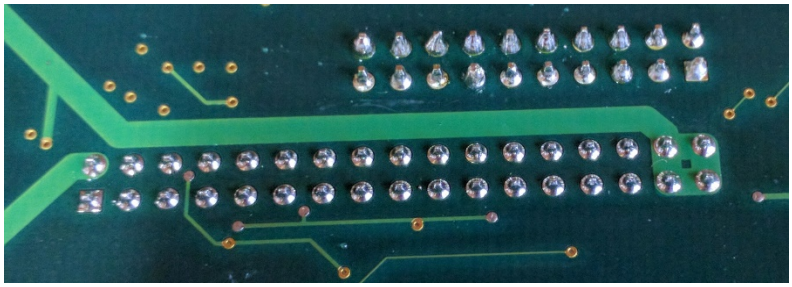
- Where are interesting areas (depopulated pads, test-points, unsure)?

  Development and debug interfaces are typically removed before production.



- How do components relate to one another?
  Observe general layout, components will be closest to what they interface with.

- How is the board powered?
  You will need to power the board. Good starting point for tracing. Big traces means big current. Good chance it is a power line.



2.2 Chip Identification

Gather information about each chip on the board for use later. Build a "Bill-of-Materials" (BOM).

- What are each of the chips and what do they do?
  Identify memory, processors, and interface controllers. The more you know the better.

2.3 Continuity Probing and Tracing

Reverse engineer the circuitry to better understand the board function and zero in on areas of interest. Draw schematics by hand as you develop an understanding during this phase.

- 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 cpu or interfaces and cpu 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.*

3. Hardware Modification (~30 minutes)

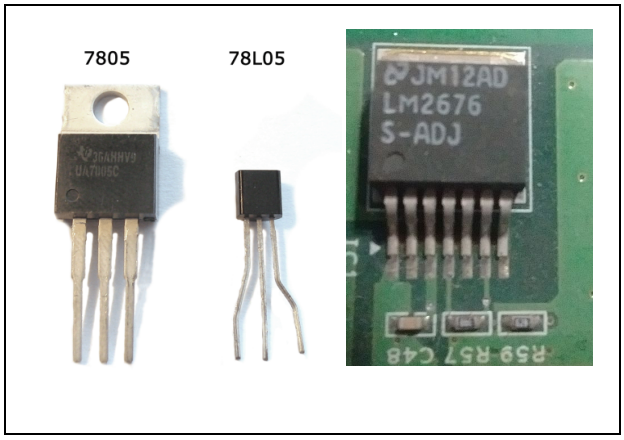
4. Active Probing (~90 minutes)

After thorough assessment with the board unpowered, it is time to attempt to provide power to the board. In industry, this is called “smoke-testing” – give the board juice and watch for smoke. If you see smoke, you’ve failed. Otherwise, you’ve passed.

In automotive, most of the time you will find 12v DC is the proper power supply because a car battery is 12v DC. However, this is not always the case. Look for stickers on the case or power supply voltage in the documentation. If you are completely unsure, start at a lower voltage and gradually work up until the board appears to be functioning properly.

4.1 Multimeter Probing

Start with the power input and use the multimeter to check voltage levels at various points on the board. Focus on voltage regulators.



Examples of regulators

- What are the voltage domains?
  Know voltage domains to interface with board later without creating smoke.

- Use to verify chip and connector pin-outs.
  Sanity check what you think you know about the components. Is GND and Vdd where you expect it to be?

- Use to find UART transmission lines or CAN lines.
  Voltage on UART TX and CAN lines will fluctuate rapidly while data is transmitted.

4.2 Oscilloscope Probing

Probing with an oscilloscope will give you a better picture of what is happening on the wire.

- Find digital buses
- Determine baud rates
- Decode signals

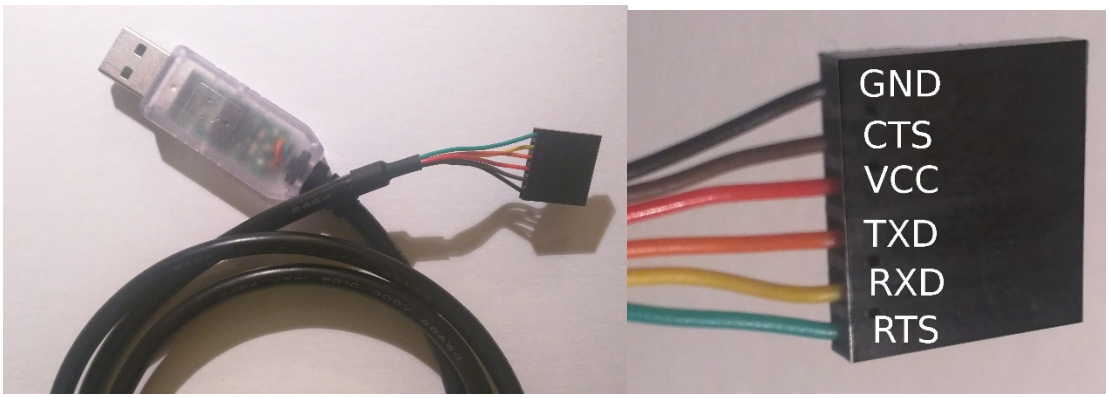
4.3 UART Probing

UART stands for “Universal Asynchronous Receiver/Transmitter.” It is a widely used serial communication protocol that HW:RE can be used to:

- Obtain an interactive serial console with a target platform
- Read system and debugging logs
- Communicate with various interfaces

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

We will be using a pretty ubiquitous “FTDI cable” as shown below.



We will also be using miniature clamp probes to attach to various places on the board to enable us to interface with the components.

4.3.1 CLASS GOALS

4.3.1.1 Find the system boot console

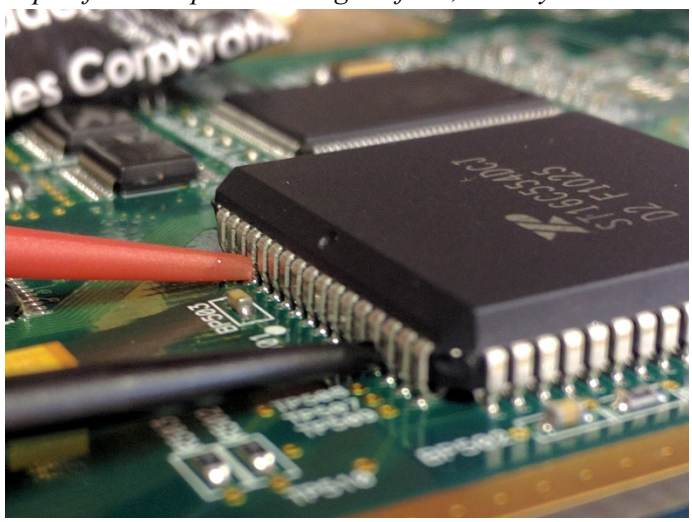
Bootstrap Version	
Memory Size	
NVRAM Init Location	
MFG Vers	

4.3.1.2 Find system debug logs

PPPC baud_rate	
PPPC data_bits	
PPPC stop_bits	
CMxx default_cc_phone	

4.3.2 Setup

1. Wire UART cable to board. Don't wire VCC.
   HINT: You are supposed to figure this part out yourself. You should already know where the UART needs to hook up to from the previous stages. If not, check your work and come back ☺



2. Install minicom:

```
sudo apt-get install minicom
```

3. 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.

4. Give permissions to /dev/ttyUSB0 and run minicom:

```
sudo chmod o+rw /dev/ttyUSB0
minicom -D /dev/ttyUSB0
```

5. Turn off hardware flow control to enable sending characters:

```
CTRL-A Z
O
“Serial port setup”
F
```

6. Useful commands
   All commands from the main window begin with CTRL-A followed by a single key press.

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

Easter Eggs:

0:00:00.000 Trace Header type 0xDEADBEEF Apps/Agents/Trace/Trace.c, (793)

0:00:03.950 CMxx Restarted ...humbly awaiting instructions...

4.4 On-Chip Debugging

JTAG (which stands for Joint Test Action Group) 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. The PeopleNet G3 Board has an NXP LH7A400 microcontroller. Inside is an ARM922T core (an ARM9). That ARM922T core supports JTAG, and, previously, we soldered a 2x10-pin header onto the board. That header is the “JTAG port.” The Bus Pirate tool has a SPI interface and is capable of interacting with the processor through the JTAG

port. In this section, we will connect the Bus Pirate to the JTAG port and interact with the processor. JTAG will allow us to dump the processor's memory and perform interactive control of the running program.

Interactive debugging is even more useful than static software reverse engineering. Whereas, during software reverse engineering on a binary file, a reverse engineer must attempt to follow the control flow to find interesting areas such as crypto operations or input processing, during interactive debugging, the processor follows the control flow itself. The reverse engineer can hitch along for the ride.

Interactive debugging can be used for more than just reverse engineering the software, though. Take, for example, a cryptographic operation performed in software using a private key. It may be very interesting for a reverse engineer to find and extract that key. By combining software reverse engineering with interactive debugging, the reverse engineer could set breakpoints in code that seems to be related to the crypto operations. If the reverse engineer finds the correct code, they may then be able to merely watch what memory addresses the processor accesses until it accesses the key. Then, the key can be extracted from memory (using the in-circuit debugging technique). This may be especially useful for obtaining ephemeral security data when analyzing an internet-connected system, as such intermediate data is not available in the software binary, but is only downloaded or produced using some information from somewhere off the victim hardware when needed.

4.4.1 Setup Bus Pirate, OpenOCD, GDB and Eclipse

The microcontroller on the PeopleNet G3 is an ARM9, which is a relatively old part. In addition, the open-source ecosystem has mature support for the ARM architecture, including the Thumb instruction set architecture (ISA), which seems to be more common in embedded systems due to reduced code size. Because of this, we can use extremely inexpensive or free tools to attach to the PeopleNet G3's JTAG port.

We will use the following tools to interface with the LH7A400 processor via the 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 – 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 – The GNU Debugger (GDB) is a free software command line software debugger.
- Eclipse – Eclipse is an open-source IDE/debugger framework, which provides a visual interface for software debugging.

Before we can begin using these tools, we will need to install and configure them.

Throughout this section, we use `/dev/ttyUSBX` as the name of the Bus Pirate device file. The actual device file name will most likely be named `/dev/ttyUSB0` or `/dev/ttyUSB1` on a GNU/Linux. You can

use `dmesg` to find which device file was created for the bus pirate after plugging it in (type 'sudo `dmesg`' in the terminal and look near the end of the log for the appropriate log entry).

4.4.1.1 Upgrade Bus Pirate firmware

Older Bus Pirate firmware versions did not support OpenOCD. Therefore, you must upgrade the Bus Pirate firmware (upgrade to v6.1 of the official firmware).

- Download the v6.1 Bus Pirate firmware and the updater script from the Bus Pirate GitHub (https://github.com/BusPirate/Bus_Pirate) and place them in the same folder.
  - https://github.com/BusPirate/Bus_Pirate/raw/master/package/BPv3-firmware/old-versions/BPv3-frimware-v6.1.hex
  - https://github.com/BusPirate/Bus_Pirate/blob/master/package/BPv3-firmware/pirate-loader_lnx?raw=true
- Connect to the Bus Pirate to the USB port of the PC
- Connect to the Bus Pirate via `screen` (or `minicom` if you prefer):
  ```
  screen /dev/ttyUSBX 115200
  ```
- If the Bus Pirate prompt doesn't appear, hit enter or press '?' (help) or 'i' (information about the device) and hit enter and the prompt should appear (possibly after some other output).
- At the Bus Pirate prompt in screen press '$' and, when prompted
  ```
  HiZ>$
  Are you sure? y
  BOOTLOADER
  ```
- Finally, we're ready to update the firmware in the Bus Pirate. Open a terminal and change the directory to where the two files in the first step where downloaded (if you don't already have one open). Set the pirate-loader_lnx file we downloaded to executable and run it (both as root):
  ```
  sudo chmod +x pirate-loader_lnx
  sudo ./pirate-loader_lnx --dev=/dev/ttyUSBX --hex=BPv3-frimware-v6.1.hex
  ```
- The pirate-loader_lnx file should output some firmware update progress information. At the end, you should see the success message:
  ```
  Firmware updated successfully :)!
  Use screen /dev/ttyUSBX 115200 to verify
  ```
- Unplug the Bus Pirate from the PC, wait a moment and reconnect the Bus Pirate. Connect one final time with screen and verify that the firmware is the correct version (in bold should match what you see):
  ```
  Bus Pirate v3.5
  Firmware v6.1 r1676  Bootloader v4.4
  DEVID:0x0447 REVID:0x3046 (24FJ64GA002 B8)
  http://dangerousprototypes.com
  ```

If you updated the Bus Pirate firmware successfully, then it's ready to be used with OpenOCD.

4.4.1.2 Install OpenOCD

OpenOCD is an open-source project for on-chip debugging. To install OpenOCD, we will simply use APT to get the version in the package manager repository:

```
sudo apt-get install openocd
```

4.4.1.3 Wire the Bus Pirate to the JTAG port

You are now ready to connect the Bus Pirate to the JTAG connector you soldered onto the PeopleNet G3 previously. The Bus Pirate documentation (http://dangerousprototypes.com/docs/Bus_Pirate_JTAG_connections_for_OpenOCD) shows which pins of the Bus Pirate connect to which pins of the JTAG header, and the ARM website has a page dedicated to the pin-out for the ARM 20-pin JTAG connector (http://infocenter.arm.com/help/topic/com.arm.doc.dui0499d/BEHEIHCE.html). Follow the Bus Pirate instructions, except connect the AUX pin of the Bus Pirate to TRST rather than SRST on the JTAG connector. The remaining connections are as follows:

```
BP   - JTAG
VPU  - VTref(3v3)
GND  - GND
MOSI - TDI
MISO - TDO
CLK  - TCK
CS   - TMS
N/A  - TRST
N/A  - RTCK
AUX  - SRST
```

4.4.1.4 Attempt to connect to board using OpenOCD attached to Bus Pirate

First, ensure that you aren't connected to `/dev/ttyUSBX` with any other program, such as `screen` or `minicom`. We'll need to add permissions to the file /dev/ttyUSB0:

```
sudo chmod o+rw /dev/ttyUSB0
```

Create two configurations, buspirate-simple.cfg:

```
source [find interface/buspirate.cfg]

buspirate_vreg 0
buspirate_mode open-drain
buspirate_pullup 1

buspirate_port /dev/ttyUSBX
```

and lh7a400.cfg :

```
reset_config trst_only separate trst_open_drain

jtag_ntrst_delay 100

set _CORE arm920t

if { [info exists CHIPNAME] } {
```

```
   set _CHIPNAME $CHIPNAME
} else {
   set _CHIPNAME lh7a400
}

if { [info exists ENDIAN] } {
   set _ENDIAN $ENDIAN
} else {
   set _ENDIAN little
}

if { [info exists CPUTAPID] } {
   set _CPUTAPID $CPUTAPID
} else {
   set _CPUTAPID 0x00922f0f
}
jtag newtap $_CHIPNAME cpu -irlen 4 -ircapture 0x1 -irmask 0xf -expected-id $_CPUTAPID

set _TARGETNAME $_CHIPNAME.cpu
target create $_TARGETNAME $_CORE -endian $_ENDIAN -chain-position $_TARGETNAME
```

Now, connect the Bus Pirate to the PeopleNet G3 JTAG connector and your computer via USB if you have not already done so. Power on the board and wait a few seconds then try to attach:

```
openocd -f buspirate-simple.cfg –f lh7a400.cfg
```

If everything is successful you will see this exact output (except for the OpenOCD version number in the first line possibly):

```
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Warn : Adapter driver 'buspirate' did not declare which transports it allows; assuming
legacy JTAG-only
Info : only one transport option; autoselect 'jtag'
srst_only separate srst_gates_jtag srst_open_drain connect_deassert_srst
trst_only separate trst_open_drain
jtag_ntrst_delay: 100
lh7a400.cpu
Info : Buspirate Interface ready!
Info : This adapter doesn't support configurable speed
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
```

4.4.1.5 Fixing the Problem

Did you receive the correct message on running OpenOCD? If not, what did you receive? Can you fix this problem and get the JTAG connection working to the PeopleNet G3?

4.4.1.6 Connect to OpenOCD over telnet

OpenOCD exposes a telnet server for interacting directly with it. To open that interface, open a telnet connection to localhost at port 4444:

```
telnet localhost 4444
```

Some interesting OpenOCD commands you might try are `targets`, `halt` and `dump_image`.

4.4.1.7 Install GDB and GCC

Now that you have OpenOCD working with the hardware, you can install GDB and Eclipse for visual debugging. As with OpenOCD, we will install GDB and GCC using apt-get

```
sudo apt-get install gcc-arm-none-eabi gdb-arm-none-eabi
```

4.4.1.8 Install and configure Eclipse

To install Eclipse, you will need to download and run the installer from the Web.

- Point Eclipse's workspace to $/tools/eclipse
  - $ is the root of the git repository
- Install the GDB Hardware Debugging plug-in
  1. Go to "**Help**" -> "**Install New Software...**"
  2. Choose the "**CDT**"... below "**—All Available Sites—**"
  3. Search for "GDB"
  4. Check the box to the left of "C/C++ GDB Hardware Debugging"
  5. Click "**Next >**" twice
  6. Select "**I accept the terms of the license agreement**"
  7. Click the "**Finish**" box
  8. Restart Eclipse when prompted

4.4.1.9 Dump the ROM memory of the NXP LH7A400

At this point, you have all the tools you need to extract a full software image from the NXP LH7A400 microcontroller on the PeopleNet G3. Using the information you have gathered and the tools you have, extract the firmware binary from the hardware.

Glossary

- **Breakpoint** – A **breakpoint** is a hardware (or software) trigger to stop the processor when performing on-chip debugging.
- **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.
- **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.
- **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.