Hardware Hacking with the Beaglebone (Bl|H)ack
● focus on reverse engineering and exploit development
● 10 years of fun with software
  ○ vuln research
  ○ security patch diffing
  ○ exploit development
  ○ security training
● Hardware Security:
  ○ medical devices, soho routers, IoT

Jeremy Richards
@dypegnosis
jeremy@0xtech.com
● Electrical Engineering education with focus on CS and Infosec
● 10 years of fun with hardware
  ○ silicon debug
  ○ security research
  ○ pen testing of CPUs
  ○ security training
● Hardware Security Training:
  ○ “Applied Physical Attacks on x86 Systems”

Joe FitzPatrick
@securelyfitz
joefitz@securinghardware.com
In the beginning, there were Vendor-supplied Proprietary tools.
Then, everyone said:

“Let’s make a low-cost, general purpose serial interface tool”
Then, everyone looked at what they had made, and it was good
But technology moves on, and there are better specialized tools for many things...
Too many are single purpose tools (also I’m messy)
How about a new all-purpose hardware hacking tool?
Why the Beaglebone Black?

- It’s cheap!
- It’s readily available
- It runs it’s own software
- It has hardware ports for:
  - UART
  - SPI
  - I2C
  - CAN
  - and more….
- It has GPIO’s and is easy to program
<table>
<thead>
<tr>
<th>Task</th>
<th>Pre-BusPirate</th>
<th>Bus Pirate</th>
<th>Post-BusPirate</th>
<th>Beaglebone Hack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Talk UART</td>
<td>$$$$</td>
<td>$$</td>
<td>$$$</td>
<td>$$</td>
</tr>
<tr>
<td>Interface I2C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dump SPI Flash</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analyze Logic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JTAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Howtos
UART FTDI Cable

One of the lightest weight method of getting a UART console is the FTDI Cable.

The cable requires drivers to be installed (windows) and creates a com port a terminal program is used to connect to the device.

Careful not to hook that red wire up to anything important ;) (RIP)
UART - Shikra & other FTDI based devices
UART - BusPirate
UART - Shikra & other FTDI based devices
UART with BBH

>echo BB-UART4 > /sys/devices/bone_capemgr.*/slots

UART4:

RX  P9_11
TX  P9_13
CTS P8_35
RTS P8_33
root@beaglebone:~# miniterm.py /dev/tty04 -b 115200
--- Miniterm on /dev/tty04: 115200,8,N,1 ---
--- Quit: Ctrl+]  | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---

U-Boot 2014.01-14400-gda781c6-dist (Apr 30 2014 - 22:35:38)

CPU:    Freescale i.MX28 rev1.2 at 454 MHz
BOOT:   NAND, 3V3
DRAM:   64 MiB
NAND:   128 MiB
In:     serial
Out:    serial
Err:    serial
Net:    FEC0 [PRIME]
Hit any key to stop autoboot: 0
UBI: attaching mtd1 to ubi0
UBI: physical eraseblock size: 131072 bytes (128 KiB)
UBI: logical eraseblock size: 126976 bytes
UBI: smallest flash I/O unit: 2048
UBI: VTD header offset: 2048 (aligned 2048)
UBI: data offset: 4096
UBI: attached mtd1 to ubi0
UBI: MTD device name: "mtd=3"
UBI: MTD device size: 8 MiB
UART - Need GPIO pins for something else?
Miniterm on /dev/tty04: 115200, 8, N, 1

Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H

U-Boot 2014.01-14400-gda781c6-dirty (Apr 30 2014 - 22:35:38)

CPU: Freescale i.MX28 rev1.2 at 454 MHz
BOOT: NAND, 3V3
DRAM: 64 MiB
NAND: 128 MiB
In: serial
Out: serial
Err: serial
Net: FEC0 [PRIME]

Hit any key to stop autoboott: 0
UBI: attaching mtd1 to ubi0
UBI: physical eraseblock size: 131072 bytes (128 KiB)
UBI: logical eraseblock size: 126976 bytes
UBI: smallest flash I/O unit: 2048
UBI: VTD header offset: 2048 (aligned 2048)
UBI: data offset: 4096
UBI: attached mtd1 to ubi0
UBI: MTD device name: "mtd=3"
UBI: MTD device size: 8 MiB
+ udhcpcd -s /etc/udhcpcd.conf
+ hciconfig hci0 up
+ bluetoothd
+ hciconfig hci0 leadv
+ sleep 2
+ touch /tmp/ap_mode
+ exit 0
Starting lighttpd: OK
Starting Zigbee...Starting lutron-core... [ OK ]
Starting aprond...Got Z-Wave version: Z-Wave 3.79
[ZWAVE OK]
/home/pi/mainsrc/MAIN/main() (Main|apron.c:51): APRON Home Automation Gateway version 1.2.0+localhost.localdomain-git[499953e]-20150410.024001 Starting ...
Starting Wink...Starting monit...hub[1145]: NOTICE: (hub.c:342) hub-dev started up by User: 0
hub[1145]: INFO: (ConfigHandler.c:98) Reading Config from: /root/config/hub.conf
hub[1145]: INFO: (hub.c:385) Waiting for /database/token
Starting monit daemon
hub[1145]: WARNING: (hub.c:416) No Token Found
hub[1145]: DEBUG: (AuthenticationUtil.c:28) Destroying Oauth
hub[1145]: DEBUG: (AuthenticationUtil.c:36) Done freeing oauth
Setting non-canonical mode
Startup complete.
+ sleep 2
+ touch /tmp/ap_mode
+ exit 0
Starting lighttpd: OK
Starting Zigbee...
Starting lutron-core... [ OK ]
Starting aprond... Got Z-Wave version: Z-Wave 3.79
[ZWAVE_OK]
i: [1139.1] main() (Main/apron.c:51): AFRON Home Automation Gateway version 1.2.0+localhost.localdomain-git{499953e-20150410.024001}
Starting ...
Starting Wink...
Starting monit... hub[1145]: NOTICE: (hub.c:342) hub-dev started up by User: 0
hub[1145]: INFO: (ConfigHandler.c:98) Reading Config from: /root/config/hub.conf
hub[1145]: INFO: (hub.c:385) Waiting for /database/token
Starting monit daemon
hub[1145]: WARNING: (hub.c:416) No Token Found
hub[1145]: DEBUG: (AuthenticationUtil.c:28) Destroying Oauth
hub[1145]: DEBUG: (AuthenticationUtil.c:36) Done freeing oauth
Setting non-canonical mode
Startup complete.

ls
+ sleep 2
+ touch /tmp/ap_mode
+ exit 0
Starting lighttpd: OK
Starting Zigbee...Starting lutron-core... [ OK ]
Starting aprond...Got Z-Wave version: Z-Wave 3.79
[ZWAVE OK]

i: [1139.1] main() (Main|apron.c:51): AFRON Home Automation Gateway version 1.2.0+localhost.localdomain-git{499953e-20150410.024001}
Starting Wink...Starting monit...hub[1145]: NOTICE: (hub.c:342) hub-dev started up by User: 0
hub[1145]: INFO: (ConfigHandler.c:98) Reading Config from: /root/config/hub.conf
hub[1145]: INFO: (hub.c:385) Waiting for /database/token
Starting monit daemon
hub[1145]: WARNING: (hub.c:416) No Token Found
hub[1145]: DEBUG: (AuthenticationUtil.c:28) Destroying Oauth
hub[1145]: DEBUG: (AuthenticationUtil.c:36) Done freeing oauth
Setting non-canonical mode
Startup complete.

ls

^C^C^C
UART - Wink Hub root

Method: Get Uboot to freak out by glitching NAND RAM. We will make the NAND flash available at first check then short it to cause the kernel image load to fail… and then drop into an interactive shell that lets us define environment variable. Copy existing and then add init=/bin/sh
UART - Wink Hub root

Method: We will make the NAND flash available at first check
Method: We will make the NAND flash available at first check...

```
U-Boot 2014.01-14400-gda781c6-dirty (Apr 30 2014 - 22:35:38)
CPU: Freescale i.MX28 rev1.2 at 454 MHz
BOOT: NAND, JTAG
DRAM: 64 MiB
NAND: 128 MiB
In: serial
Out: serial
Err: serial
Net: FECO [PRIME]
Hit any key to stop autoboot: 0
UBI: attaching mtd to ubi0
UBI: physical eraseblock size: 131072 bytes (128 Kibi)
UBI: logical eraseblock size: 126976 bytes
UBI: smallest flash I/O unit: 2048
UBI: VID header offset: 2048 (aligned 2048)
UBI: data offset: 4096
UBI: attached mtd to ubi0
UBI: MTD device name: "mtd=3"
UBI: MTD device size: 8 MiB
UBI: number of good PEBs: 64
UBI: number of bad PEBs: 0
UBI: max. allowed volumes: 128
```
UART - Wink Hub root

then short it to cause the kernel image load to fail…
...and then drop into an interactive shell that lets us define environment variable.
UART - Wink Hub root

Method: Copy existing app_bootargs= and then add init=/bin/sh. Finally run the app_boot (yellow)
UART - Wink Hub root

Method: Copy existing `app_bootargs=` and then add `init=/bin/sh`. Finally run the `app_boot` (yellow)
UART - Wink Hub root

Method: Copy existing app_bootargs= and then add init=/bin/sh. Finally run the app_boot (yellow)
<table>
<thead>
<tr>
<th>Task</th>
<th>Pre-BusPirate</th>
<th>Bus Pirate</th>
<th>Post-BusPirate</th>
<th>Beaglebone Hack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Talk UART</td>
<td>RS232 hardware + level shifting</td>
<td>narrow tolerance</td>
<td>FT232R, just works, $$</td>
<td>native hardware</td>
</tr>
<tr>
<td>Interface I2C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dump SPI Flash</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analyze Logic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JTAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
I2C

I2C  AKA I$^2$C
AKA IIC  AKA eye-two-see
AKA eye-squared-see  AKA aye-eye-see

Also, SMBus, 2-wire, and much more are similar in concept and often compatible...
And hasn’t updated their website since...
And hasn’t updated their website since...
2 I2C ports

<table>
<thead>
<tr>
<th>P9</th>
<th>1</th>
<th>2</th>
<th>P8</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGND</td>
<td>1</td>
<td>2</td>
<td>DGND</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>VDD_3V3</td>
<td>3</td>
<td>4</td>
<td>VDD_3V3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>VDD_5V</td>
<td>5</td>
<td>6</td>
<td>VDD_5V</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>SYS_5V</td>
<td>7</td>
<td>8</td>
<td>SYS_5V</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>PWR_BUT</td>
<td>9</td>
<td>10</td>
<td>SYS_RESETN</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>GPIO_30</td>
<td>11</td>
<td>12</td>
<td>GPIO_60</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>GPIO_31</td>
<td>13</td>
<td>14</td>
<td>GPIO_40</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>GPIO_48</td>
<td>15</td>
<td>16</td>
<td>GPIO_51</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>I2C1_SCL</td>
<td>17</td>
<td>18</td>
<td>I2C1_SDA</td>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>I2C2_SCL</td>
<td>19</td>
<td>20</td>
<td>I2C2_SDA</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>I2C2_SCL</td>
<td>21</td>
<td>22</td>
<td>I2C2_SDA</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>GPIO_49</td>
<td>23</td>
<td>24</td>
<td>I2C1_SCL</td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>GPIO_117</td>
<td>25</td>
<td>26</td>
<td>I2C1_SDA</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>GPIO_125</td>
<td>27</td>
<td>28</td>
<td>GPIO_123</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>GPIO_121</td>
<td>29</td>
<td>30</td>
<td>GPIO_122</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>GPIO_120</td>
<td>31</td>
<td>32</td>
<td>VDD_ADC</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>AIN4</td>
<td>33</td>
<td>34</td>
<td>GND_ADC</td>
<td>33</td>
<td>34</td>
</tr>
<tr>
<td>AIN6</td>
<td>35</td>
<td>36</td>
<td>AIN5</td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td>AIN2</td>
<td>37</td>
<td>38</td>
<td>AIN3</td>
<td>37</td>
<td>38</td>
</tr>
<tr>
<td>AIN0</td>
<td>39</td>
<td>40</td>
<td>AIN1</td>
<td>39</td>
<td>40</td>
</tr>
<tr>
<td>GPIO_20</td>
<td>41</td>
<td>42</td>
<td>GPIO_7</td>
<td>41</td>
<td>42</td>
</tr>
<tr>
<td>DGND</td>
<td>43</td>
<td>44</td>
<td>DGND</td>
<td>43</td>
<td>44</td>
</tr>
<tr>
<td>DGND</td>
<td>45</td>
<td>46</td>
<td>DGND</td>
<td>45</td>
<td>46</td>
</tr>
</tbody>
</table>
root@beaglebone:/home/debian# echo BB-I2C1 > /sys/devices/bone_capemgr.*/slots
root@beaglebone:/home/debian# i2cdetect -l
i2c-0  i2c  OMAP I2C adapter                   I2C adapter
i2c-1  i2c  OMAP I2C adapter                   I2C adapter
i2c-2  i2c  OMAP I2C adapter                   I2C adapter
root@beaglebone:/home/debian# i2cdetect -r 2
WARNING! This program can confuse your I2C bus, cause data loss and worse!
I will probe file /dev/i2c-2 using read byte commands.
I will probe address range 0x03-0x77.
Continue? [Y/n]
   0 1 2 3 4 5 6 7 8 9 a b c d e f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- 14 -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
<table>
<thead>
<tr>
<th>Task</th>
<th>Pre-BusPirate $$$$</th>
<th>Bus Pirate $$</th>
<th>Post-BusPirate $$$</th>
<th>Beaglebone Hack $$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Talk UART</td>
<td>RS232 hardware + level shifting</td>
<td>narrow tolerance</td>
<td>FT232R, just works, $$</td>
<td>native hardware</td>
</tr>
<tr>
<td>Interface I2C</td>
<td>?</td>
<td>passable</td>
<td>Aardvark/Beagle - $$$</td>
<td>native hardware</td>
</tr>
<tr>
<td>Dump SPI Flash</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analyze Logic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JTAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SPI

Serial Peripheral Interface

- FLASH chips
- SD Cards
- Sensors
- Displays
- more...

>echo BB-SPIDEV0 > /sys/devices/bone_capemgr.*/slots

>flashrom -p linux_spi:dev=/dev/spidev1.0 -r dumpfile.bin
| buspirate_spi:dev=/dev/ttyUSB0,spispeed=1M |
|---|---|---|---|---|---|---|---|
| **HiZ** | **1wire** | **UART** | **I2C 2wire** | **SPI 3wire** | **JTAG** | **LA** |
| MISO | RX | MISO | TDO | 1 |
| CS/TMS | OWD | TX | SDA | MOSI | TDI | 3 |
| MOSI/TX | SCL | SCK | TCK | 2 |
| AUX | AUX I/O | -PWM | -Measures Hz (5Vmax) | 4 |
| Vpu | Input Pullup Resistors (0-5V) |
| ADC | Analog/Digital converter (6Vmax) |
| 5V | 5V | 5V | 5V | 5V | 5V | 5V |
| 3V3 | 3V3 | 3V3 | 3V3 | 3V3 | 3V3 |
| GND | GND | GND | GND | GND | GND | GND |
2 SPI ports

<table>
<thead>
<tr>
<th>P9</th>
<th>P8</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DGND</strong></td>
<td><strong>DGND</strong></td>
</tr>
<tr>
<td>VDD_3V3</td>
<td>VDD_3V3</td>
</tr>
<tr>
<td>VDD_5V</td>
<td>VDD_5V</td>
</tr>
<tr>
<td>SYS_5V</td>
<td>SYS_5V</td>
</tr>
<tr>
<td>PWR_BUT</td>
<td>SYS_RESETN</td>
</tr>
<tr>
<td>GPIO_30</td>
<td>GPIO_60</td>
</tr>
<tr>
<td>GPIO_31</td>
<td>GPIO_40</td>
</tr>
<tr>
<td>GPIO_48</td>
<td>GPIO_51</td>
</tr>
<tr>
<td>SPI0_CS0</td>
<td>SPI0_D1</td>
</tr>
<tr>
<td>SPI1_CS0</td>
<td>SPI1_D0</td>
</tr>
<tr>
<td>SPI0_D0</td>
<td>SPI0_SCLK</td>
</tr>
<tr>
<td>GPIO_49</td>
<td>GPIO_15</td>
</tr>
<tr>
<td>GPIO_117</td>
<td>GPIO_14</td>
</tr>
<tr>
<td>GPIO_125</td>
<td>SPI1_CS0</td>
</tr>
<tr>
<td>SPI1_D0</td>
<td>SPI1_D1</td>
</tr>
<tr>
<td>SPI1_SCLK</td>
<td>VDD_ADC</td>
</tr>
<tr>
<td>AIN4</td>
<td>GND_ADC</td>
</tr>
<tr>
<td>AIN6</td>
<td>AIN5</td>
</tr>
<tr>
<td>AIN2</td>
<td>AIN3</td>
</tr>
<tr>
<td>AIN0</td>
<td>AIN1</td>
</tr>
<tr>
<td>GPIO_20</td>
<td>SPI1_CS1</td>
</tr>
<tr>
<td>DGND</td>
<td>DGND</td>
</tr>
<tr>
<td>DGND</td>
<td>DGND</td>
</tr>
</tbody>
</table>

Joe FitzPatrick & Jeremy Richards
spi on the BBH
# echo BB-SPIDEV0 > /sys/devices/bone_capemgr.*/slots
# time flashrom -p linux_spi:dev=/dev/spidev1.0 -r dumpfile.bin

flashrom v0.9.8-r1888 on Linux 3.8.13-bone47 (armv7l)
flashrom is free software, get the source code at http://www.flashrom.org

Calibrating delay loop... OK.
Found Spansion flash chip "S25FL208K" (1024 kB, SPI) on linux_spi.

Reading flash... done.

real  0m2.616s
user  0m0.900s
sys   0m0.168s
<table>
<thead>
<tr>
<th>Task</th>
<th>Pre-BusPirate $$ $$ $$,  ⬅ ⬅ ⬅</th>
<th>Bus Pirate $$,  ⬅ ⬅ ⬅</th>
<th>Post-BusPirate $$ $$ $$,  ⬅ ⬅</th>
<th>Beaglebone Hack $$,  ⬅</th>
</tr>
</thead>
<tbody>
<tr>
<td>Talk UART</td>
<td>RS232 hardware + level shifting</td>
<td>narrow tolerance</td>
<td>FT232R, just works, $$</td>
<td>native hardware</td>
</tr>
<tr>
<td>Interface I2C</td>
<td>?</td>
<td>passable</td>
<td>Aardvark/Beagle - $$ $$ $$</td>
<td>native hardware</td>
</tr>
<tr>
<td>Dump SPI Flash</td>
<td>Universal Programmer $$ $$ $$</td>
<td>slow  ⬅ ⬅ ⬅ ⬅ ⬅ ⬅ ⬅ ⬅ ⬅</td>
<td>ft232H, $$ teensy/arduino $$</td>
<td>native hardware insanely fast</td>
</tr>
<tr>
<td>Analyze Logic</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JTAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
logic analyzer - beaglelogic
logic analyzer - beaglelogic

BeagleLogic turns your BeagleBone [Black] into a 14-channel, 100Msps Logic Analyzer. Once loaded, it presents itself as a character device node /dev/beaglelogic.

- 'beaglelogic' kernel module
- two Programmable Real-Time Units (PRUs)
- works with the sigrok library

https://github.com/abhishek-kakkar/BeagleLogic
logic analyzer - sigrok

>`echo BB-BEAGLELOGIC > /sys/devices/bone_capemgr.*/slots`

>`modprobe beagelogic`

>`echo 33554432 > /sys/devices/virtual/misc/beaglelogic/memalloc`
logic analyzer - sigrok

Basic raw captures with dd

>dd if=/dev/beaglelogic of=mydump bs=1M count=1

sigrok support

>sigrok-cli --time 10s -o test-capture-1.sr -d beaglelogic -c samplerate=500khz --channels P8_45,P8_46
Rendered in 57 ms.

This is a static test waveform.

Rendering... This may take a couple of seconds, and make the browser window non-responsive. Please be patient!
logic analyzer - 12 (+2) chan

exclusive-use =

//  "P8.20",    /* pru1: pr1_pru1_pru_r31_13 */
//  "P8.21",    /* pru1: pr1_pru1_pru_r31_12 */
"P8.27",    /* pru1: pr1_pru1_pru_r31_8 */
"P8.28",    /* pru1: pr1_pru1_pru_r31_10 */
"P8.29",    /* pru1: pr1_pru1_pru_r31_9 */
"P8.30",    /* pru1: pr1_pru1_pru_r31_11 */
"P8.39",    /* pru1: pr1_pru1_pru_r31_6 */
"P8.40",    /* pru1: pr1_pru1_pru_r31_7 */
"P8.41",    /* pru1: pr1_pru1_pru_r31_4 */
"P8.42",    /* pru1: pr1_pru1_pru_r31_5 */
"P8.43",    /* pru1: pr1_pru1_pru_r31_2 */
"P8.44",    /* pru1: pr1_pru1_pru_r31_3 */
"P8.45",    /* pru1: pr1_pru1_pru_r31_0 */
"P8.46",    /* pru1: pr1_pru1_pru_r31_1 */
logic analyzer - sigrok protocol decoders

> sigrok-cli -i test-capture-2.sr -P uart:baudrate=115200:parity_type=none -B uart

Above is a UART example. sigrok can also decode CAN (automotive), i2c, JTAG, modbus, 1wire, parallel, sdcard spi, spi flash, SWD, USB packet

A full list of protocols with decoders is available here:

http://sigrok.org/wiki/Protocol_decoders
<table>
<thead>
<tr>
<th>Task</th>
<th>Pre-BusPirate $$$, 🏆🏆🏆</th>
<th>Bus Pirate $, 🏆🏆🏆</th>
<th>Post-BusPirate $$$, 🏆</th>
<th>Beaglebone Hack $$, 🏆</th>
</tr>
</thead>
<tbody>
<tr>
<td>Talk UART</td>
<td>RS232 hardware + level shifting</td>
<td>narrow tolerance</td>
<td>FT232R, just works, $$</td>
<td>native hardware</td>
</tr>
<tr>
<td>Interface I2C</td>
<td>?</td>
<td>passable</td>
<td>Aardvark/Beagle - $$$</td>
<td>native hardware</td>
</tr>
<tr>
<td>Dump SPI Flash</td>
<td>Universal Programmer $$$</td>
<td>slow 🏆🏆🏆🏆🏆</td>
<td>ft232H, $$ teensy/arduino $$</td>
<td>native hardware insanely fast 🏆</td>
</tr>
<tr>
<td>Analyze Logic</td>
<td>Benchtop equipment $$$</td>
<td>limited capture</td>
<td>saleae $$</td>
<td>native hardware</td>
</tr>
<tr>
<td>JTAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
JTAG - Work in Progress

OpenOCD has a driver for toggling GPIO via Sysfs:

```plaintext
interface sysfsgpio
# Each of the JTAG lines need a gpio number set: tck tms tdi tdo
# Header pin numbers: ## ## ## ##

sysfsgpio_jtag_nums ## ## ## ##
# At least one of srst or trst needs to be specified

# Header pin numbers: TRST - ##, SRST - ##
sysfsgpio_trst_num ##
sysfsgpio_srst_num ##
```
JTAG - Work in Progress

To use it:

# echo BB-JTAG > /sys/devices/bone_capemgr.*/slots
# openocd -f sysfsgpio-bbb.cfg -f target.cfg
<table>
<thead>
<tr>
<th>Task</th>
<th>Pre-BusPirate $$$, 🌟🌟🌟🌟</th>
<th>Bus Pirate $$, 🌟🌟🌟🌟🌟</th>
<th>Post-BusPirate $$$, 🌟🌟</th>
<th>Beaglebone Hack $$, 🌟🌟🌟</th>
</tr>
</thead>
<tbody>
<tr>
<td>Talk UART</td>
<td>RS232 hardware + level shifting</td>
<td>narrow tolerance</td>
<td>FT232R, just works, $$</td>
<td>native hardware</td>
</tr>
<tr>
<td>Interface I2C</td>
<td>?</td>
<td>passable</td>
<td>Aardvark/Beagle - $$$</td>
<td>native hardware</td>
</tr>
<tr>
<td>Dump SPI Flash</td>
<td>Universal Programmer $$$$$</td>
<td>slow 🌟🌟🌟🌟🌟🌟🌟🌟🌟</td>
<td>ft232H, $$ teensy/arduino $$</td>
<td>native hardware insanely fast 🌟🌟🌟🌟🌟🌟🌟🌟🌟</td>
</tr>
<tr>
<td>Analyze Logic</td>
<td>Benchtop equipment $$$$$</td>
<td>limited capture</td>
<td>saleae $$$</td>
<td>native hardware</td>
</tr>
<tr>
<td>JTAG</td>
<td>Vendor-supplied $$$$$</td>
<td>flakey</td>
<td>ft232h $$</td>
<td>GPIO via sysfs perf. like ft232h</td>
</tr>
</tbody>
</table>
Beaglebone Capes

Allow expandability onto the BBB

Have an EEPROM so they’re auto-detected

GPIOs are configured automatically!
Why a cape?

It’s nice to have clearly labeled headers for UART, SPI, JTAG, etc…

It’s nice to buffer your I/O so you don’t kill your BBB

It’s really nice to have level shifting to let us use 1.8 to 5.5 on our pins!
Design Decisions

BBB I/O is 3.3v

It’s NOT 5v tolerant - wires we poke around with should be

Level Shifting up OR down is pretty straightforward

But for this part (and MANY others):

\[ V_{CCA} \leq V_{CCB} \]

We can’t have \( V_{CCA}=3.3V \) and \( V_{CCB}=1.8V \) to 5V
Suitable Parts

SN74LVC8T245 and SN74LVCH16T245:

$V_{CCA}$: A-port supply voltage. $1.65 \, V \leq V_{CCA} \leq 5.5 \, V$

$V_{CCB}$: B-port supply voltage. $1.65 \, V \leq V_{CCB} \leq 5.5 \, V$
Suitable Parts

SN74LVC8T245 and SN74LVCH16T245:

$V_{CCA}$: A-port supply voltage. $1.65 \text{ V} \leq V_{CCA} \leq 5.5 \text{ V}$

$V_{CCB}$: B-port supply voltage. $1.65 \text{ V} \leq V_{CCB} \leq 5.5 \text{ V}$

Bidirectional - but direction has to be set
Suitable Parts

SN74LV4T125:

Up Translation
1.2 V\(^{(1)}\) to 1.8 V at 1.8-V V\(_{CC}\)
1.5 V\(^{(1)}\) to 2.5 V at 2.5-V V\(_{CC}\)
1.8 V\(^{(1)}\) to 3.3 V at 3.3-V V\(_{CC}\)
3.3 V to 5.0 V at 5.0-V V\(_{CC}\)

Down Translation
3.3 V to 1.8 V at 1.8-V V\(_{CC}\)
3.3 V to 2.5 V at 2.5-V V\(_{CC}\)
5.0 V to 3.3 V at 3.3-V V\(_{CC}\)

Unidirectional - but single supply!
Suitable Parts

Bidirectional up and down solution?

Does not seem to exist :(

We could:

- Shift up to 5v, then down to 1.8-5
- Have separate Up and Down translation
- Translate down, and protect the inputs with zener diodes
Design in Development

SN74LVCH16T245
for Beaglelogic and all output-only signals

SN74LV4T125
for input-only signals

TXS0102
for bidirectional signals, with zener diodes
<table>
<thead>
<tr>
<th>Task</th>
<th>Pre-BusPirate</th>
<th>Bus Pirate</th>
<th>Post-BusPirate</th>
<th>Beaglebone Hack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Talk UART</td>
<td>RS232 hardware + level shifting</td>
<td>narrow tolerance</td>
<td>FT232R, just works, $$</td>
<td>native hardware</td>
</tr>
<tr>
<td>Interface I2C</td>
<td>?</td>
<td>passable</td>
<td>Aardvark/Beagle - $$$</td>
<td>native hardware</td>
</tr>
<tr>
<td>Dump SPI Flash</td>
<td>Universal Programmer $$$</td>
<td>slow $</td>
<td>ft232H, $$ teensy/arduino $$</td>
<td>native hardware insanely fast 🙄</td>
</tr>
<tr>
<td>Analyze Logic</td>
<td>Benchtop equipment $$$</td>
<td>limited capture</td>
<td>saleae $$$</td>
<td>native hardware</td>
</tr>
<tr>
<td>JTAG</td>
<td>Vendor-supplied $$ $$</td>
<td>flakey</td>
<td>ft232h $$</td>
<td>GPIO via sysfs perf. like ft232h</td>
</tr>
</tbody>
</table>
Future Ideas

Facedancer functionality (USB MITM) - This is partially working as part of the USB proxy project (https://github.com/dominicgs/USBProxy)

BusPirate emulation over GPIO

Sigrok cloud decoding (REST web service)

JTAG identification via GPIO + Logic Analyzer/Decoders
Final Tips

If you’re getting unexpected output you may not have enough power. The BeagleBone can be powered by 5v power adapter if it’s not getting enough power over USB.

If your wires are long you might get some strange results when dumping (eg spi). You can increase stability by reducing the speed.

... -p linux_spi:dev=/dev/spidev1.0,spispeed=1000 ...

spispeed is in khz, so 1000 =1mhz
Final Tips

Know what bone firmware you are running. Configuring the tools differs between versions.

Have a second one… or 10. It is the best way to test/debug/do a sanity check.
Q&A