Raspberry Pi Armduino Controller

Design Documentation

uControl, Inc.	Document No.	700-1072-001
5790 Manuel Place	Revision	A
Sugar Hill, GA 30518	Date	26 October 2015
dmcneill@icloud.com	Author	Don McNeill

Date	Author	Revision	Changes
26 October 2015	Don McNeill	Initial Publication	Initial Publication

Table of Contents

Table of Figures

Table of Listings

Table of Tables

Scope

This document describes the design of the ARM Controller Board add-on option for the embedded Raspberry Pi Revision B+/2 platform. The goal of the design is to provide a easy to use ARM Cortex-M3 micro controller for embedded software product development and for general embedded experimentation.

Overview

Recent advances in ARM Cortex architecture in the past few years have made it feasible to couple an additional micro controller to the Raspberry PI in a robust manner. One core feature is the ability to download a program over the serial interface from the Raspberry Pi to the micro controller by using the RESET and ISP signals in conjunction with the resident on-board micro controller firmware. Another feature is the access to the micro controller's digital port pins and exposing the I2C interface and SPI.

The operation of the Armduino microcontroller can be attached to a Raspberry Pi B+ or Raspberry Pi B 2 board. It is also possible to run the Armduino microcontroller in a stand-alone mode by itself where power is obtained through the USB connector. The Armduino also has a set of interface connectors that inplement the Arduino UNO interface. The Arduino UNO connectors can be used for both +3.3V and +5V boards by selection of the voltage at the jumper J1:

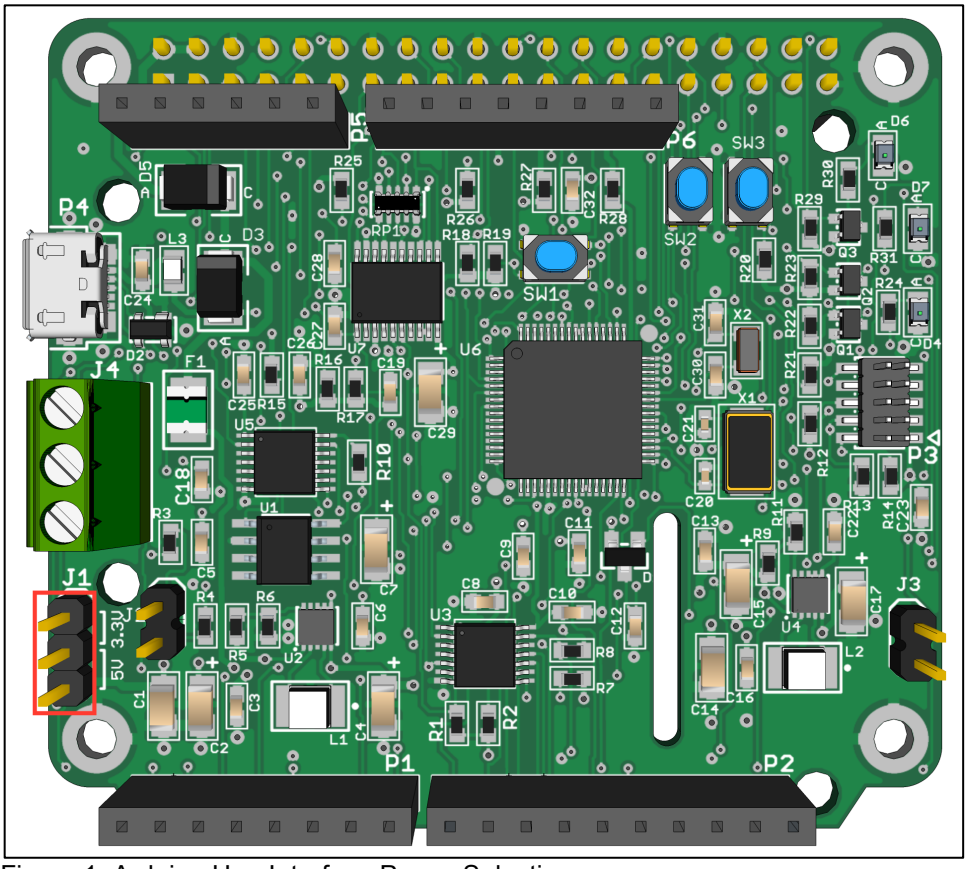

Figure 1: Arduino Uno Interface Power Selection

It is also possible to cascade devices from the Raspberry Pi to attach additional Armduino controllers. If the USB is enabled on the Armduino, the secondary USB device is /dev/ttyACM0…n. The main Armduino (the one attached to the Raspberry Pi) has a device name of /dev/ttyAMA0. This makes it possible for the Raspberry Pi to control multiple Armduino boards through the virtual USB serial port:

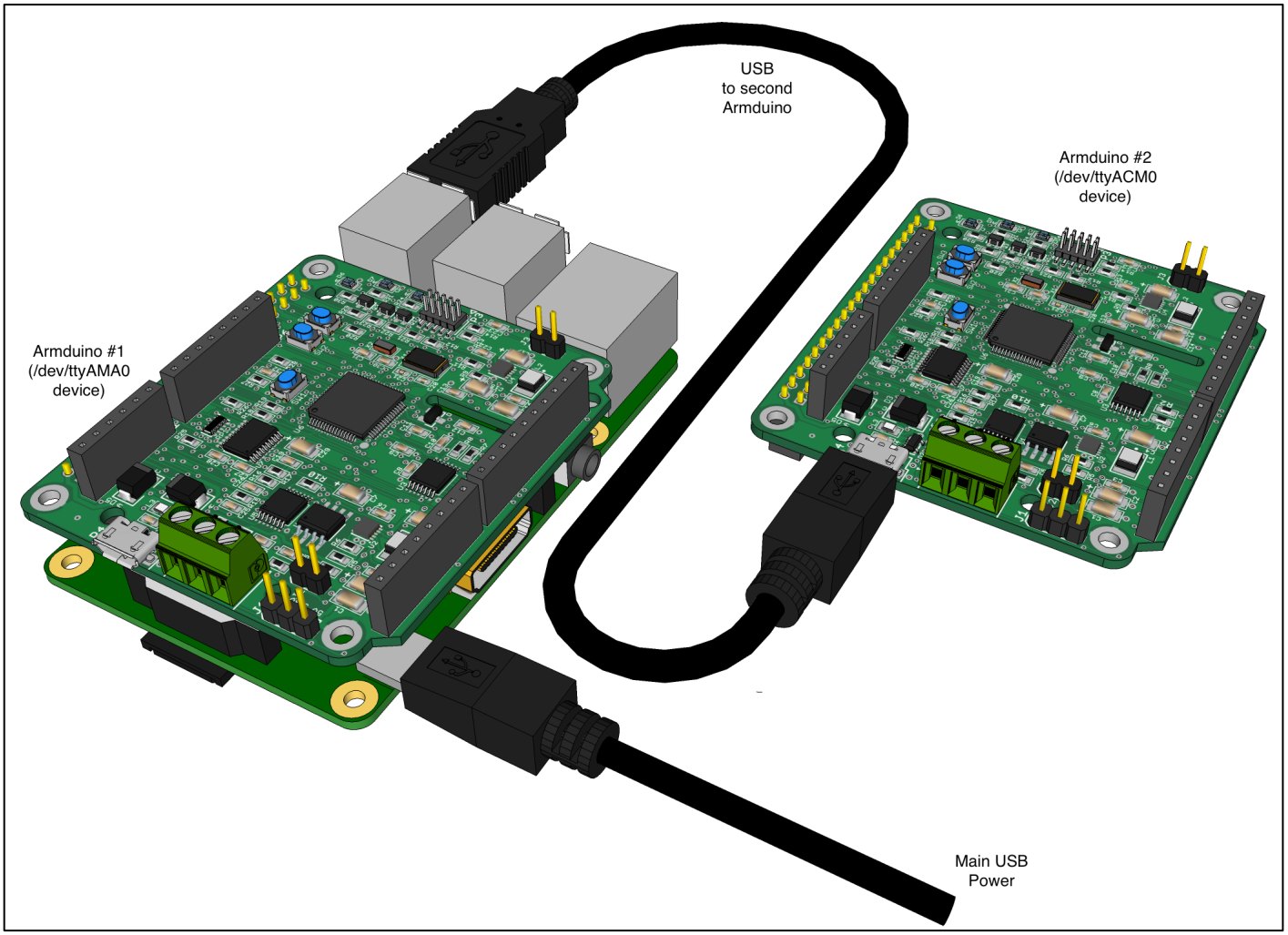

Figure 2: Cascading Armduino Boards from Raspberry Pi

Design

The design of the Armduino uses the Raspberry Pi as a base for controlling the Armduino, as detailed below in the block diagram:

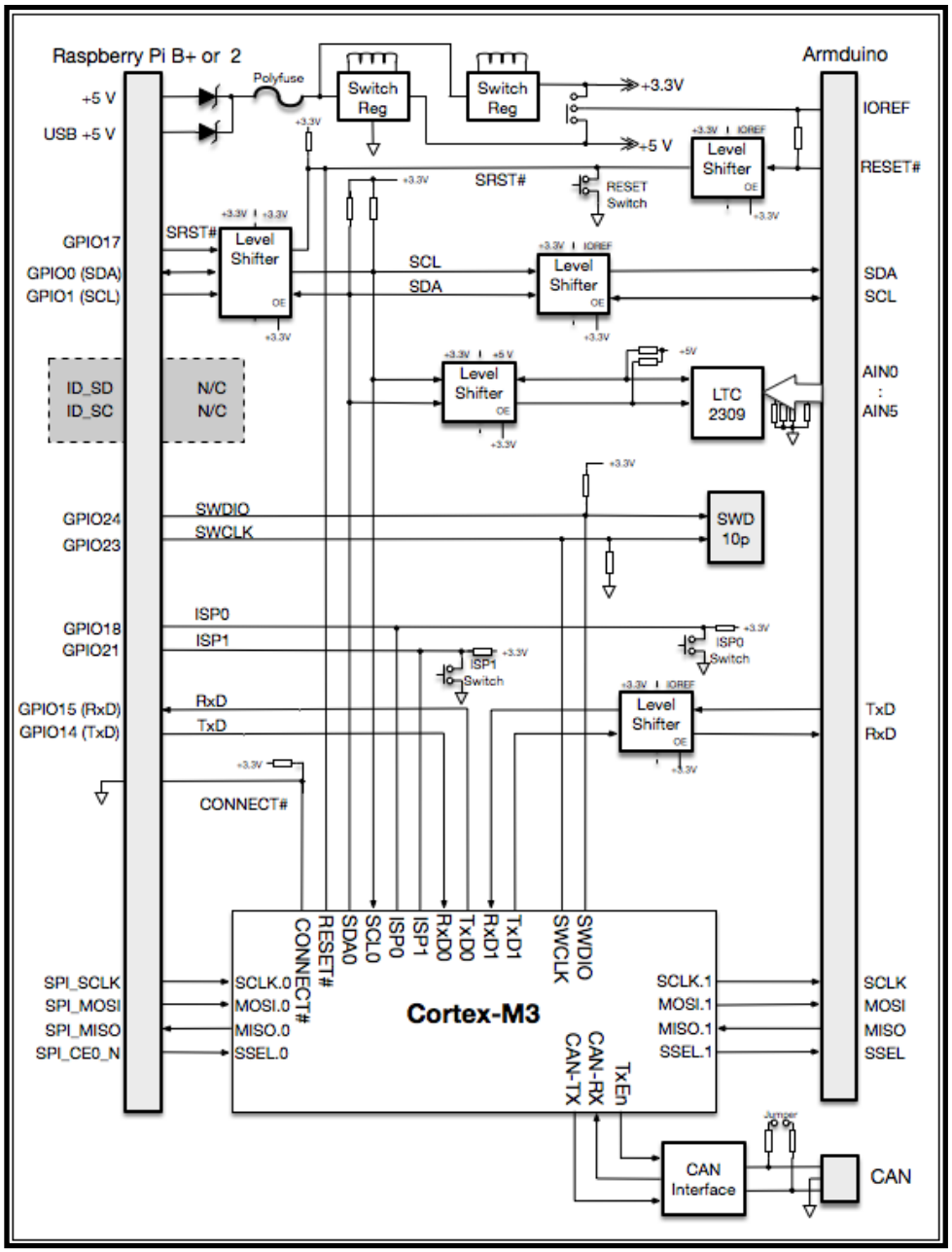

Figure 3: Armduino Block Diagram

Signal Description

The following signals are used in this design:

Raspberry Pi			Armduino	
Pin	Signal Name	Direction	Signal Name	Function
1	+3.3V	Power	VCC3V3-NC	+3.3Volt Power [NOT USED]
2	+5V	Power	VCC5V-RPI	+5 Volt Power
3	GPIO 0 (SDA)	I/O	SDA0	General I/O or I2C SDA I/O
4	+5V	Power	VCC5V-RPI	+5 Volt Power
5	GPIO 1 (SCL)	I/O	SCL0	General I/O or I2C SCL Output
6	GND	Ground	GND	Ground Signal
7	GPIO 4	I/O	GPIO_GCLK	
8	GPIO 14 (TXD)	Out	RXD-1549	Serial TxD
9	GND	Ground	GND	Ground Signal
10	GPIO 15 (RXD)	I/O	TXD-1549	Serial RxD
11	GPIO 17	Out	SRST#	RESET signal to micro controller.
12	GPIO 18 (PCM_CLK)	Out	ISP_0	ISP signal to micro controller.
13	GPIO 21 (PCM_DOUT)	Out	ISP_1	ISP_SEL signal to micro controller.
14	GND	Ground	GND	Ground Signal
15	GPIO 22	I/O	GPIO 22	General Purpose I/O
16	GPIO 23	Out	SW_CLK	Used for SWD.
17	+3.3V	Power	VCC3V3-RPI	+3.3Volt Power [NOT USED]
18	GPIO 24	Out	SWDIO	Used for SWD.
19	GPIO 10 (MOSI)	I/O	MOSI	General I/O or SPI MOSI
20	GND	Ground	GND	Ground Signal
21	GPIO 9 (MISO)	I/O	MISO	General I/O or SPI MISO
22	GPIO 25	I/O	GPIO_GEN6	RED LED on the Armduino
23	GPIO 11 (SCLK)	I/O	SCLK0	General I/O or SPI SCLK
24	GPIO 8 (CE0)	I/O	SSEL0	General I/O or SPI CE0 Output
25	GND	Ground	GND	Ground Signal
26	GPIO 7 (CE1)	I/O	SSEL1	General I/O or SPI CE1 Output
27	ID_SD	NC	Not-Connected	Not-Used

Raspberry Pi			Armduino	
Pin	Signal Name	Direction	Signal Name	Function
28	ID_SC	NC	Not-Connected	Not-Used
29	GPIO 5	I/O		General Purpose I/O
30	GND	Ground	CONNECT#	Detect connection to Raspberry Pi
31	GPIO 6	I/O		General Purpose I/O
32	GPIO 12	I/O		General Purpose I/O
33	GPIO 13	I/O		General Purpose I/O
34	GND	Ground	GND	Ground Signal
35	GPIO 19	I/O		General Purpose I/O
36	GPIO 16	I/O		General Purpose I/O
37	GPIO 26	N/C	Not-Connected	Not-Used
38		N/C	Not-Connected	Not-Used
39	GND	Ground	GND	Ground Signal
40		N/C	Not-Connected	Not-Used

Table 1: Armduino - Raspberry Pi Signals

Microcontroller Control Signals

The signals that control the state of the microcontroller are ISP_0, ISP_1 and RESET. The RESET signal is inverted to allow the power up of the Raspberry Pi to assert reset to the microcontroller. The ISP_SEL selects the “In-System Programmable” mode, whether it is through the serial port, ‘UART’, through the CAN interface or through the USB interface. Normally, both ISP signals should be set to the logic high state (No ISP).

Signal	Function
ISP_1,0	LH = CAN Mode for programming. HL = USB Mode for programming. LL = USART Mode for programming. HH = Normal Operation (Running)
RESET	Reset: L = Normal Operation (Running), H = Reset State
SWCLK	GPIO for SWCLK; normally inactive as an input pin.
SWDIO	GPIO for SWDIO; normally inactive as an input pin.

Table 2: In-System Programming Signals

Microcontroller Interface Signals

The other signals that interface to the microcontroller are as follows:

Signal	Interface	Function
SDA	I2C	I2C Data Signal
SCL	I2C	I2C Clock Signal
SPI_SCK	SPI	SPI Clock Signal
SPI_MOSI	SPI	SPI Master-Out Slave-In Data Signal
SPI_MISO	SPI	SPI Master-In Slave-Out Data Signal
SPI_CE0	SPI	SPI Chip Select 0
GEN_CPIO6	I/O	General Interface SIgnal
TxD	UART	UART (Serial) Interface
RxD	UART	UART (Serial) Interface

Table 3: Armduino Interface Signals

The microcontroller is capable of interfacing to the Raspberry PI on the I2C and the SPI interface. On the microcontroller side, the SPI interface has the characteristic for CPHA = 0 where the SPI Chip Select must be set inactive between successive frames. The Raspberry Pi hardware transmits successive frames continually with no intervening break on the SPI Chip Select between frames. Therefore, the Raspberry Pi should be set to Mode 1 or Mode 3, which corresponds to SPI CPHA = H (logic one). The modes are shown in the table below:

SPI Mode	CPOL	CPHA
0	L	L
1	L	H
2	H	L
3	H	H

Table 4: SPI Modes

The I2C signals SCL and SDA from the Armduino microcontroller are gated through a bi-directional port to interface to the Raspberry Pi. Normally, the enable signal is active-high, the default value on power-up is a logic one, high signal due to the pull-up on the enable signal. If the microcontroller detects the CONNECT# signal to be high, then the microcontroller can gate the I2C_ENABLE signal to the inactive state, thus turning off access to the I2C port at the Raspberry Pi connector. Normally, this is only done in stand-alone mode, where the Armduino microcontroller is operating by itself without the Raspberry Pi connected.

Armduino Signals

The signals available at the Armduino for control / signaling purposes are shown in the table below:

Table 5: Armduino Signals

Port/Bit	Signal Name	Source	Destination	Function
PIO_0.0	SSEL1	P100.26	U6.2	SPI Select #1 from Raspberry Pi
PIO_0.1	CONNECT#	P100.30	U6.5	Connect Signal to Armduino
PIO_0.2	I2C_ENABLE	U6.6	U5.8	Enable I2C Bridge from Armduino/Raspberry Pi to Arduino
PIO_0.3	GPIO4	P100.7	U6.7	GPIO Signal 4 from Raspberry Pi
PIO_0.4	GPIO25	P100.22	U6.8	Red LED Indicator signal from Raspberry Pi
PIO_0.5	IO.2	U6.9	P1.6	Arduino IO.2 Signal to Armduino
PIO_0.6	GPIO22	P100.15	U6.10	GPIO Signal 22 from Raspberry Pi
PIO_0.7	CAN_TX_EN	U6.11	U1.8	Enable for CAN Bus Interface
PIO_0.8	CAN-Rx	U6.12	U1.4	CAN-Rx TTL-Level Signal
PIO_0.9	SCLK0	P100.23	U6.16	SPI Master Clock from Raspberry Pi
PIO_0.10	MISO	P100.21	U6.19	SPI Master-In-Slave-Out from Raspberry Pi
PIO_0.11	MOSI	P100.19	U6.23	SPI Master-Out-Slave-In from Raspberry Pi
PIO_0.12	SSEL0	P100.24	U6.24	SPI Select #0 from Raspberry Pi
PIO_0.13	RXD-1549	P100.8	U6.29	UART Serial RxD to Raspberry Pi
PIO_0.14	IO.4	U6.30	P1.4	Arduino IO.4 Signal to Armduino
PIO_0.15	IO.5	U6.31	P1.3	Arduino IO.5 Signal to Armduino
PIO_0.16	IO.6	U6.32	P1.2	Arduino IO.6 Signal to Armduino
PIO_0.17	IO.3	U6.39	P1.5	Arduino IO.3 Signal to Armduino
PIO_0.18	TXD-1549	P100.10	U6.17	UART Serial TxD to Raspberry Pi
PIO_0.19	SWCLK	P100.16	U6.40	SWD Clock Signal from Raspberry Pi
PIO_0.20	SWDIO	P100.18	U6.44	SWD IO Signal from Raspberry Pi
PIO_0.21	RESET#	P100.11	U6.45	Active-Low Reset from Raspberry Pi
PIO_0.22	SCL	U6.49	P2.1	I2C Bus Clock Signal from Armduino to Arduino
PIO_0.23	SDA	U6.50	P2.2	I2C Bus Data Signal from Armduino to Arduino
PIO_0.24	X.SCK	U6.58	P2.5	SPI Master Clock from Armduino to Arduino
PIO_0.25	X.MISO	U6.60	P2.6	SPI Master-In-Slave-Out from Armduino to Arduino
PIO_0.26	X.MOSI	U6.61	P2.7	SPI Master-Out-Slave-In from Armduino to Arduino
PIO_0.27	X.SSEL	U6.62	P2.8	SPI Select from Armduino to Arduino
PIO_0.28	IO.7	U6.63	P1.1	Arduino IO.7 Signal to Armduino
PIO_0.29	IO.8	U6.64	P2.10	Arduino IO.8 Signal to Armduino
PIO_0.30	IO.9	U6.1	P2.9	Arduino IO.9 Signal to Armduino
PIO_0.31	CAN-TX	U6.3	U1.1	CAN-Tx TTL-Level Signal
PIO_1.0	GPIO5	P100.29	U6.4	GPIO Signal 5 from Raspberry Pi
PIO_1.1	GPIO6	P100.31	U6.15	GPIO Signal 6 from Raspberry Pi
PIO_1.2	GPIO12	P100.32	U6.25	GPIO Signal 12 from Raspberry Pi
PIO_1.3	GPIO13	P100.33	U6.28	GPIO Signal 13 from Raspberry Pi

PIO_1.4	GPIO16	P100.36	U6.33	GPIO Signal 16 from Raspberry Pi
PIO_1.5	GPIO19	P100.35	U6.34	GPIO Signal 19 from Raspberry Pi
PIO_1.6	TXD-ARD	U6.46	P1.7	UART Serial TxD to Arduino
PIO_1.7	RXD-ARD	U6.51	P1.8	UART Serial RxD from Arduino
PIO_1.8	LED1	U6.53	Q2.GATE	Green LED Indicator #1
PIO_1.9	ISP_0	P100.12	U6.54	Arduino In-System-Programmable Mode Signal 0 from Raspberry Pi
PIO_1.10	LED2	U6.59	Q1.GATE	Green LED Indicator #2
PIO_1.11	ISP_1	P100.13	U6.38	Arduino In-System-Programmable Mode Signal 1 from Raspberry Pi

Raspberry Pi Installation

To control the Armduino from the Raspberry Pi, the Pi must have the following options setup through the raspi-config utility:

```
1)    Expand Filesystem
3)    Boot Options
          -> B1 Console
8)    Advanced Options
          -> A2 Hostname
               -> raspberrypi
          -> A4 SSH
               -> Enable
          -> A4 Device Tree
               -> Yes
               -> Ok
          -> A7 I2C
               -> Yes
               -> Ok
               -> Yes
               -> Ok
          -> A8 Serial
               -> No
               -> Ok
          -> A9 Audio
               -> 0 Auto
Finish
Yes
```

Listing 1: Commands for 'raspi-config'

To control the Armduino from the Raspberry Pi, there are some script files that should be downloaded from the following repository:

git@github.com:dmcneill/RaspberryPi-Armduino.git

This can be cloned on the Raspberry Pi using the following commands:

```
sudo apt-get install minicom
mkdir -p Projects
cd Projects
git clone https://github.com/dmcneill/RaspberryPi-Armduino.git
cd RaspberryPi-Armduino
sudo chmod +x rc.* armduino minirc.dfl
sudo chown root.root rc.* armduino minirc.dfl
sudo cp rc.* /etc
sudo cp minirc.dfl /etc/minicom
sudo cp armduino /etc/init.d
sudo update-rc.d armduino defaults
sudo service armduino start
sudo usermod -aG gpio pi
```

Listing 2: Raspberry Pi Setup Commands

The other item is to add a line to the 'halt' script at /etc/init.d/halt:

```
NETDOWN=yes

PATH=/sbin:/usr/sbin:/bin:/usr/bin

[ -f /etc/default/halt ] && . /etc/default/halt

[ -f /etc/rc.teardown ] && . /etc/teardown

. /lib/lsb/init-functions
```

Listing 3: /etc/init.d/halt Script Modification

The change to the /etc/init.d/halt script is to invoke the /etc/rc.teardown script if that script file exists. This is to force the reset to be asserted and teardown all the other signals.

The intent for the GPIO export is to be able to programmatically access the individual signals to control the Armduino from the Raspberry Pi. The GPIO access is through the system GPIO interface. This allows Linux to set the direction, bit and value for the GPIO. Note that the script *rc.function* contains the functions used in the *rc.gpio* script. For the *rc.gpio* script, any argument sets the bits used for SWD, specifically SWDIO and SWCLK, to inputs for use with an external SWD cable to P3 (the 5x2 pin connector at the back of the board).

The second file, *rc.function*, defines the functions used in the *rc.gpio* script:

```
#!/bin/sh
#
# rc.function
#
# This script defines functions for setting up GPIO on the Armduino

# Function for GPIO bit manipulation
setup() {
  local bit="$1"
  local dir="$2"
  local val="$3"
  if [ -d /sys/class/gpio/gpio${bit} ]; then
    echo $bit > /sys/class/gpio/unexport
  fi
  echo $bit > /sys/class/gpio/export
  echo $dir > /sys/class/gpio/gpio${bit}/direction
  if [ "$dir" = "out" ]; then
    echo $val > /sys/class/gpio/gpio${bit}/value
  fi
  chown root.gpio /sys/class/gpio/gpio${bit}/value
  chmod 660 /sys/class/gpio/gpio${bit}/value
}

# Function for GPIO output
output() {
  local bit="$1"
  local val="$2"
  echo $val > /sys/class/gpio/gpio${bit}/value
}

# Function for GPIO teardown
teardown() {
  local bit="$1"
  local val="$2"
  echo $val > /sys/class/gpio/gpio${bit}/value
  echo $bit > /sys/class/gpio/unexport
}
```

Listing 4: Script Function Definitions

The last file, rc.gpio, calls the setup function to set the direction, bit and output value for the GPIO bit.

```
#!/bin/bash
#
# rc.gpio
#
# This script is for setting up the GPIOs for use with the Armduino.

source /etc/rc.function

#--------------------------------------
# Setup GPIO for -RST and -ISP signals
#--------------------------------------

# -RST --> Active
setup 17 out 0

# -ISP0 --> Application
setup 18 out 1

# -ISP1 --> Application
setup 27 out 1
if [ "$1" ]; then
    # SWCLK --> Inactive
    setup 23 in 1

    # SWDIO --> Inactive
    setup 24 in 1
else
    # SWCLK --> Inactive
    setup 23 out 0

    # SWDIO --> Inactive
    setup 24 out 1
fi
# LED --> Inactive
setup 25 out 1

# -RST --> Inactive
output 17 1
```

Listing 5: FIle rc.gpio Listing

Another method to control the GPIO is the use of the WiringPi library and associated applications for setting GPIO individually. The details on this library may be obtained from http://wiringpi.com. A simple test would be to query all the GPIO pins through the WiringPi GPIO application:

```
 +-----+-----+---------+------+---+--B Plus--+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 |   IN | 1 |  3 || 4  |   |      | 5V      |     |     |
 |   3 |   9 |   SCL.1 |   IN | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 0 | IN   | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | IN   | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 1 | 11 || 12 | 1 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 1 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 1 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 1 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI |   IN | 1 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 1 | 21 || 22 | 1 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK |   IN | 1 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 1 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 1 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 1 | 35 || 36 | 1 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 1 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+--B Plus--+---+------+---------+-----+-----+
```

Listing 6: GPIO Program Output

The final script, rc.teardown, is called when the system is halted:

```
#!/bin/bash
#
# rc.teardown
#
# This script is for tearing down the GPIOs for use with the Armduino.

source /etc/rc.function

# -RST --> Active
output 17 0
teardown 18 1
teardown 27 1
teardown 23 0
teardown 24 1
teardown 25 0
exit 0
```

Listing 7: GPIO Teardown Script

The final script, armduino, is to start and stop the GPIO settings on system startup and shutdown. The script is installed by the following commands:

```
sudo chmod +x /etc/init.d/armduino
sudo update-rc.d armduino defaults
sudo service armduino start|stop|restart
```

The listing for the armduino script is shown below:

```
#! /bin/sh
# Starts and stops armduino
# /etc/init.d/armduino
### BEGIN INIT INFO
# Provides: armduino
# Required-Start:   $syslog
# Required-Stop:    $syslog
# Default-Start:    2 3 4 5
# Default-Stop:     0 1 6
# Short-Description:    Start / stop Armduino
### END INIT INFO

#Load up Armduino when called
case "$1" in

start)
    echo "Starting Armduino.."
    /etc/rc.gpio
;;

stop)
    echo "Stopping Armduino.."
    /etc/rc.teardown
;;

restart)
    echo "Restarting Armduino.."
    $0 stop
    $0 start
;;
*)
    echo "Usage: $0 {start|stop|restart}"
    exit 1
esac
```

Listing 8: The Armduino Startup / Shutdown Script

Application Development

To facilitate application development on the Armduino, download the LPCXpresso toolkit from

https://www.lpcware.com/lpcxpresso/download

Choose the installation image that fits the operating system environment; Windows, Linux or MAC OS X. Once installed, the package LPCOpen for the LPC1549 needs to be installed. Go to the website:

https://www.lpcware.com

Figure 4: LPCWare Software Download

Select the LPC15xx package by clicking on the “Downloads” text - this will bring up the download page. Select the “Software” tab, as in the figure above.

Now click on the line below the ‘Software’ tab with the text ‘LPCOpen Software Development Platform (LPC15xx)’. This will bring up another page as follows:

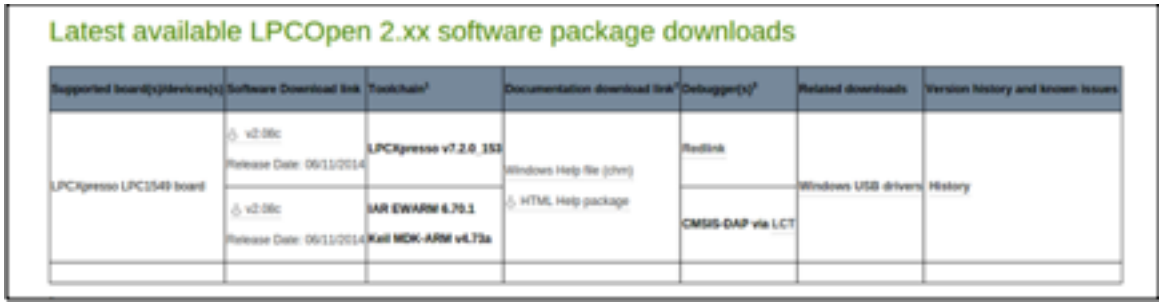

Figure 5: LPCWare Software Selection

Under the ‘Latest available LPCOpen’, select the LPCXpresso version at the top. The download will start automatically and download the zip file containing the LPCOpen package for the LPC15xx microcontroller.
Next, start up the LPCxpresso application (that was installed under the operating system of choice). On the lower left, there is a “Quickstart” panel that contains several entries. Click on the “Import project(s)” entry:

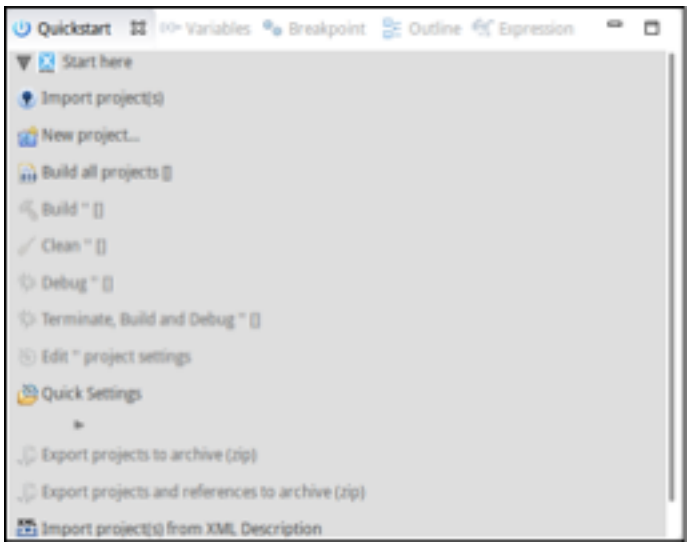

Figure 6: LPCXpresso QuickStart Pane

A dialog box will show up ‘Import Project(s)”; enter in the zip file package to be imported, it should be something like this:

/home/…/Downloads/lpcopen_2_08c_lpcxpresso_nxp_lpcxpresso_1549.zip

Click the ‘Next’ button at the bottom of the dialog. On the next dialog, a list of projects that will be imported will be shown. Make sure that all the projects are selected, then click on the ‘Finish’ button. The Project Explorer pane will show all the imported projects. Select the ‘periph_blinky’ project:

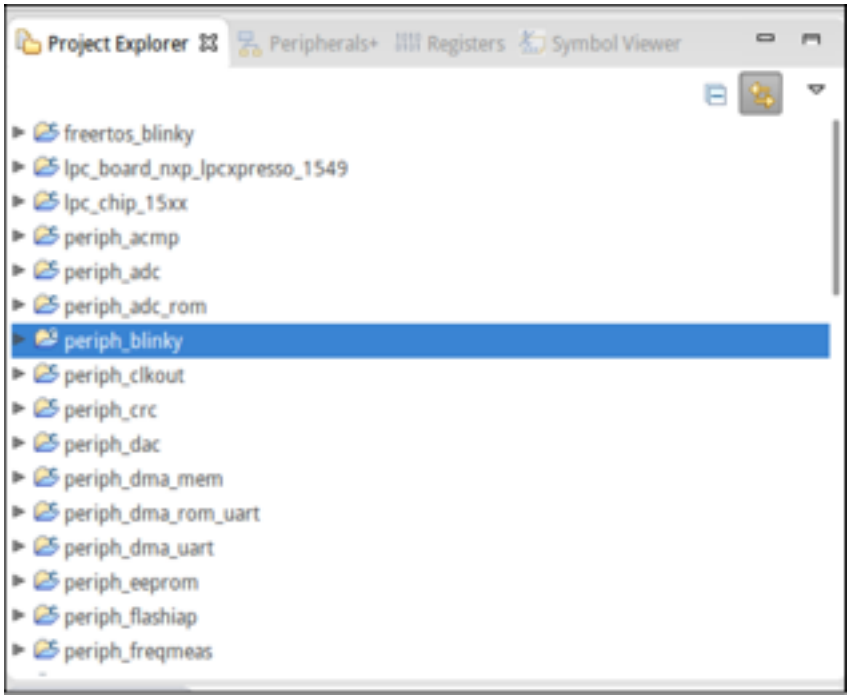

Figure 7: Project Explorer Pane

Now select the “Project->Build Project” menu - this will force the “periph_blinky” project to be built. In the file under pc_board_nxp_lpcxpresso_1549 -> src -> board.c change the following:

```
#define MAXLEDS 3
static const uint8_t ledpins[MAXLEDS] = {25, 3, 1};
static const uint8_t ledports[MAXLEDS] = {0, 0, 1};
```

Listing 9: File 'board.c' LED Original Definitions

```
#define MAXLEDS 3
static const uint8_t ledpins[MAXLEDS] = {25, 3, 1};
static const uint8_t ledports[MAXLEDS] = {0, 0, 1};
```

Listing 10: File 'board.c' LED Armduino Definitions

Save the changes. We have just modified the source code definition for the LEDs on the board to match the hardware. Select the “Project->Build Project” menu - this will force the “periph_blinky” project to be rebuilt.

Armduino Application Programming

To flash the board, there are two methods to use. The first method is to use an executable program on the Raspberry Pi to access the Armduino microcontroller in In-System-Program (ISP) mode.

In-System-Programming (ISP)

The In-System-Programming executable is named 'isp15xx'; the source code can be obtained from:

https://github.com/dmcneill/isp15xx.git

To clone the repository, please enter the following commands:

```
mkdir -p Projects
cd Projects
git clone https://github.com/dmcneill/isp15xx.git isp15xx
cd isp15xx
make clean all
sudo make install
```

Listing 11: Commands for isp15xx Installation

The command line arguments are as follows:

```
ISP Client for LPC15xx Microcontroller
Usage:
isp15xx [OPTIONS] -p -d <device> -f <filename>
isp15xx [OPTIONS] --program -device=<device> -filename=<filename>
 where:
  --erase    | -e    Erase the flash
  --program  | -p    Program the flash
  --device   | -d    Serial port device name
  --filename | -f    Intel Hex filename
 OPTIONS:
  --reset    | -r    Mark reset as active HIGH
  --nogpio   | -g    Don't use GPIO for RST, ISP
  --verbose  | -v    Verbose messages
  --examine  | -x    Examine memory
  --help     | -h    Show this help
```

Listing 12: Executable 'isp15xx' Command Line Arguments

To copy the executable from the host to the Raspberry Pi, enter in the following commands:

```
scp periph_blinky.axf pi@raspberrypi:/home/pi
```

Now on the Raspberry Pi, enter in the following commands:

```
isp15xx –p –d /dev/ttyAMA0 –f periph_blinky.axf
```

The output is as follows:

```
isp15xx -p -d /dev/ttyAMA0 -f ./periph_blinky.axf
2015-10-12 04:50:55 [INFO]:      Entering fileWorker...
2015-10-12 04:50:55 [INFO]:      filename is periph_blinky.axf  n is 13
2015-10-12 04:50:55 [INFO]:      Start: 0x00000000  End: 0x00001b1b
2015-10-12 04:50:55 [INFO]:      Updating checksum from 0x00000000 to 0xfdffb31a
2015-10-12 04:50:55 [INFO]:      CHECKSUM is 0xfdffb31a
2015-10-12 04:50:55 [INFO]:      Sectors: start=0 End:=1 count=2
2015-10-12 04:50:55 [INFO]:      Leaving fileWorker: result is 0
2015-10-12 04:50:55 [INFO]:      Entering programClient()
2015-10-12 04:50:56 [INFO]:      Baud rate set to 115200 and number of stop bits is 1
2015-10-12 04:50:56 [INFO]:      Device is 0x1549
2015-10-12 04:50:56 [INFO]:      Blank check...
2015-10-12 04:50:56 [INFO]:      Sector 0 is NOT-BLANK
2015-10-12 04:50:56 [INFO]:      Sector 1 is NOT-BLANK
2015-10-12 04:50:56 [INFO]:      Programming flash...
2015-10-12 04:50:57 [INFO]:      Writing flash at 0x00001c00
2015-10-12 04:50:58 [INFO]:      Writing flash at 0x00001800
2015-10-12 04:50:59 [INFO]:      Writing flash at 0x00001400
2015-10-12 04:51:00 [INFO]:      Writing flash at 0x00001000
2015-10-12 04:51:02 [INFO]:      Writing flash at 0x00000c00
2015-10-12 04:51:03 [INFO]:      Writing flash at 0x00000800
2015-10-12 04:51:04 [INFO]:      Writing flash at 0x00000400
2015-10-12 04:51:05 [INFO]:      Writing flash at 0x00000000
2015-10-12 04:51:05 [INFO]:      Programming flash success!
2015-10-12 04:51:05 [INFO]:      Leaving programClient(): errorCode is 0
2015-10-12 04:51:06 [INFO]:      Leaving clientWorker: result is 0
2015-10-12 04:51:06 [INFO]:      Tearing down...
```

Listing 13: Program 'isp15xx' Output Listing

Note that the isp15xx program will automatically calculate the proper checksum so that the application program that was flashed will start up. The isp15xx program will accept file types of elf, axf, bin and ihex (Intel hex) formats. The source code can be downloaded from:

git@github.com:dmcneill/isp15xx.git

LPCXpresso Flash Programming

A second method to program the Armduino microcontroller is to go through a SWD adaptor to the SWD connector P3 on the board:

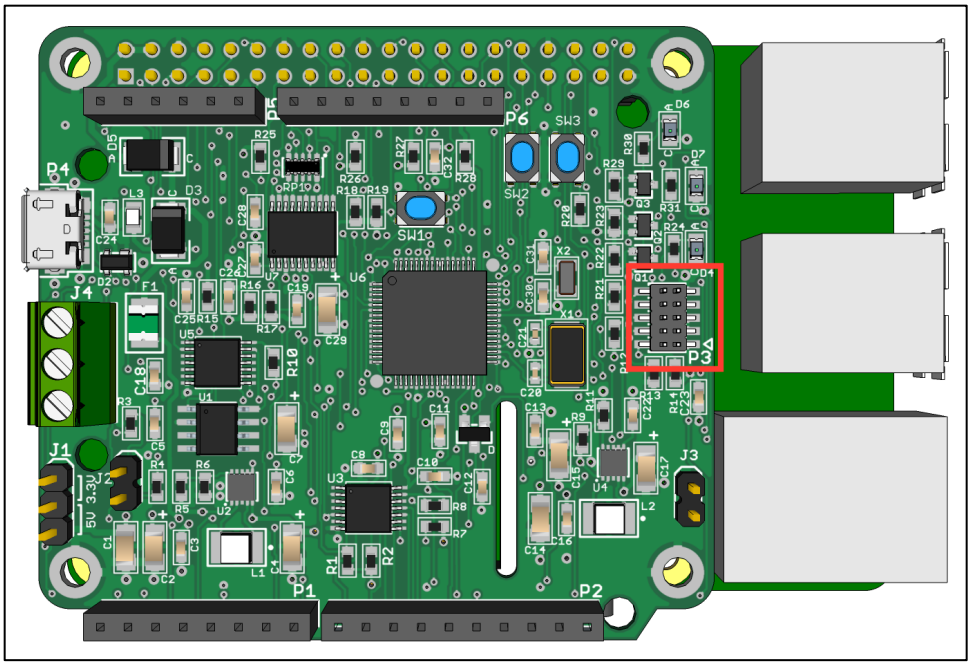

Figure 8: Armduino SWD Connector

In order to use the SWD hardware interface, the GPIO pins for SWCLK and SWDIO need to be set as inputs. In order to accomplish this, the file /etc/rc.gpio needs to be invoked with an argument (of any value) to set the SWCLK and SWDIO signals as inputs.

Armduino Application Debugging

The Armduino application can be debugged remotely using the openocd application. The source tree for this specific version of openocd for Armduino can be downloaded from:

git clone git@github.com:dmcneill/openocd.git

OpenOCD Build and Installation

The build and installation instructions host is on the Raspberry Pi itself. Install the following packages in order to build OpenOCD:

```
git clone git@github.com:dmcneill/openocd.git
sudo apt-get update
sudo apt-get install -y automake libusb-1.0.0 libudev-dev
sudo apt-get install -y autoconf libtool libftdi-dev
```

Listing 14: Package Installation for OpenOCD

Next, the HidAPI package needs to be downloaded and installed:

```
git clone git://github.com/signal11/hidapi.git
cd hidapi
./bootstrap
./configure
make
sudo make install
cd ..
```

Listing 15: HidAPI Package Installation

Now the OpenOCD package itself can be downloaded and setup:

```
git clone git@github.com:dmcneill/openocd.git && cd openocd
{ ./bootstrap && ./configure --enable-sysfsgpio --enable-bcm2835gpio \
 --enable-maintainer-mode --disable-werror --enable-ep93xx --enable-
at91rm9200 \
 --enable-usbprog --enable-jlink --enable-vsllink --enable-rlink \
 --enable-stlink --enable-arm-jtag-ew --enable-dummy --enable-buspirate \
 --enable-ulink --prefix=/usr && make; } > openocd_build.log 2>&1
sudo make install
sudo usermod -aG gpio pi
```

Listing 16: OpenOCD Installation Setup and Build

Remote Debugging

Once the build and installation for OpenOCD has finished, the OpenOCD may be used to setup a debug session directly to the Armduino board through the remote GDB interface:

```
sudo /etc/rc.gpio && openocd -s /usr/share/openocd -f board/raspberrypi-armduino.tcl
```

The output should look like the following:

```
Open On-Chip Debugger 0.10.0-dev-00040-gd52070c-dirty (2015-10-14-21:56)
Licensed under GNU GPL v2
For bug reports, read
       http://openocd.org/doc/doxygen/bugs.html
SysfsGPIO nums: swclk = 23, swdio = 24
SysfsGPIO num: srst = 17
srst_only separate srst_gates_jtag srst_open_drain connect_deassert_srst
adapter speed: 10 kHz
adapter_nsrst_delay: 200
cortex_m reset_config sysresetreq
Info : SysfsGPIO JTAG/SWD bitbang driver
Info : SWD only mode enabled (specify tck, tms, tdi and tdo gpios to add
JTAG mode)
Warn : gpio 17 is already exported
Warn : gpio 23 is already exported
Warn : gpio 24 is already exported
Info : This adapter doesn't support configurable speed
Info : SWD IDCODE 0x2ba01477
Info : lpc15xx.cpu: hardware has 6 breakpoints, 4 watchpoints
```

Listing 17: OpenOCD Output for Armduino

To debug the example program *periph_blinky*, on a Linux based computer, enter in the following commands:

```
cd ~/workspace/test/periph_blinky/Debug
/usr/local/lpcxpresso_7.9.0_455/lpcxpresso/tools/bin/arm-none-eabi-gdb ./periph_blinky.axf
```

The gdb executable will now startup on the host. Connect to the target by entering the 'target remote' command with the IP address for the Raspberry Pi on port 3333:

```
(gdb) target remote 10.0.1.88:3333
Remote debugging using 10.0.1.88:3333
0x030000b8 in ?? ()
(gdb) monitor reset init
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x030000b8 msp: 0x02001fe0
(gdb) b main
Breakpoint 1 at 0x35a: file ../example/src/systick.c, line 72.
(gdb) c
Continuing.
Note: automatically using hardware breakpoints for read-only addresses.

Breakpoint 1, main () at ../example/src/systick.c:72
72            SystemCoreClockUpdate();
(gdb) list 60
55       * @brief    Handle interrupt from SysTick timer
56       * @return   Nothing
57       */
58      void SysTick_Handler(void)
59      {
60            Board_LED_Toggle(0);
61            Board_LED_Toggle(1);
62      }
63
64      /**
(gdb) b 60
Breakpoint 2 at 0x344: file ../example/src/systick.c, line 60.
(gdb) c
Continuing.

Breakpoint 2, SysTick_Handler () at ../example/src/systick.c:60
60            Board_LED_Toggle(0);
(gdb) c
Continuing.

Breakpoint 2, SysTick_Handler () at ../example/src/systick.c:60
60            Board_LED_Toggle(0);
(gdb) d 2
(gdb) c
Continuing.
```

Listing 18: Debug Session on Armduino

References

LPC1549:
http://www.nxp.com/products/microcontrollers/product_series/lpc1500/

LPCXpresso:
https://www.lpcware.com/lpcxpresso

LPCOpen:
https://www.lpcware.com/content/nxpfile/lpcopen-platform

Raspberry Pi:
https://www.raspberrypi.org

OpenOCD:
http://openocd.org