BeagleBone Black - Complete Hardware Verification Guide

Purpose

This document explains **step-by-step** how to verify SPI, UART, and I2C interfaces after booting a BeagleBone Black with a fresh Debian image, and how to interpret the results for driver development.

Table of Contents

---

1. Initial System Check

1.1 Check Kernel Version

```
uname -r
```

What it shows:

- Kernel version running on your BBB
- Important for checking compatible drivers and features

Example Output:

```
6.12.32-bone28
```

Interpretation:

- `6.12.32` = Linux kernel version
- `bone28` = BeagleBoard-specific patch version

1.2 Check System Information

```
sudo beagle-version
```

What it shows:

- EEPROM information
- Board model
- Boot source (SD card or eMMC)
- Bootloader version
- Device tree used

- Installed packages status
- Kernel boot parameters

Key Information to Note:

- model:[TI_AM335x_BeagleBone_Black] - Confirms board type
- UBOOT: Booted Device-Tree:[am335x-boneblack.dts] - Device tree file used
- bootloader:[microSD-(push-button)] - Boot source

---

2. Verify I2C Interfaces

2.1 List I2C Device Files

```
ls /dev/i2c*
```

What it does:

- Lists all I2C character device files in /dev/
- Each file represents an I2C bus accessible from userspace

Example Output:

```
/dev/i2c-0  /dev/i2c-2
```

Interpretation:

- /dev/i2c-0 = I2C bus 0 (exists)
- /dev/i2c-2 = I2C bus 2 (exists)
- **Missing** /dev/i2c-1 means I2C1 is not enabled or not configured

Why this matters:

- If you see /dev/i2c-X, the I2C controller is enabled
- You can write drivers that use these buses
- Userspace tools can access these buses

2.2 Check I2C in Kernel Messages

```
dmesg | grep -i i2c
```

What it does:

- Searches kernel boot messages for I2C-related information
- Shows driver initialization, detected hardware, and errors

Example Output:

```
[    2.033921] i2c_dev: i2c /dev entries driver
[    3.212405] omap_i2c 4819c000.i2c: bus 2 rev0.11 at 100 kHz
[    5.189598] omap_i2c 44e0b000.i2c: bus 0 rev0.11 at 400 kHz
```

Interpretation:

- i2c_dev: i2c /dev entries driver = I2C character device driver loaded
- omap_i2c 4819c000.i2c: bus 2 rev0.11 at 100 kHz
    - **omap_i2c** = TI OMAP I2C driver

    - **4819c000.i2c** = Hardware base address (from device tree)
    - **bus 2** = This hardware is I2C bus 2
    - **rev0.11** = Hardware revision
    - **at 100 kHz** = I2C clock speed
- 44e0b000.i2c: bus 0 = I2C0 at address 0x44E0B000

How to use this:

- Hardware addresses match AM335x Technical Reference Manual
- You need these addresses when writing drivers
- Clock speed tells you the bus configuration

2.3 Find I2C Hardware Controllers

```
ls /sys/bus/platform/devices/ | grep i2c
```

What it does:

- Lists platform devices registered with the kernel
- Platform devices are memory-mapped peripherals

Example Output:

```
44e0b000.i2c
4819c000.i2c
```

Interpretation:

- 44e0b000.i2c = I2C0 controller at physical address 0x44E0B000
- 4819c000.i2c = I2C2 controller at physical address 0x4819C000

How to map to pins:

- I2C0 (0x44E0B000) = Internal, used for cape EEPROM
- I2C2 (0x4819C000) = Maps to P9.19 (SCL) and P9.20 (SDA)

Why these addresses matter:

- Your driver's probe() function receives these addresses
- They're defined in the device tree
- Match AM335x TRM Chapter 21 (I2C section)

2.4 Install and Use I2C Tools

```
# Install tools
sudo apt-get install i2c-tools

# List I2C buses
i2cdetect -l

# Scan bus 2 for devices
sudo i2cdetect -y -r 2
```

What i2cdetect -l shows:

```
i2c-0   i2c          OMAP I2C adapter                I2C adapter
i2c-2   i2c          OMAP I2C adapter                I2C adapter
```

What i2cdetect -y -r 2 does:

- Scans all addresses (0x03-0x77) on I2C bus 2
- Shows which addresses respond (device present)
- Useful for finding connected I2C peripherals

Example output:

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: 50 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

- 50 = Device found at address 0x50 (often EEPROM)
- -- = No device at this address
- UU = Address in use by kernel driver

---

3. Verify UART Interfaces

3.1 List UART Device Files

ls /dev/ttyS* /dev/ttyO*

What it does:

- Lists serial port device files
- ttyS* = Standard 8250/16550 serial ports
- ttyO* = OMAP-specific serial ports (older kernels)

Example Output:

/dev/ttyS0 /dev/ttyS1 /dev/ttyS2 /dev/ttyS3 /dev/ttyS4 /dev/ttyS5

Interpretation:

- 6 UART devices available (ttyS0 through ttyS5)
- ttyS0 = Usually debug console (don't use for testing)
- ttyS1-5 = Available for your drivers/applications

3.2 Check UART in Kernel Messages

dmesg | grep -i uart
dmesg | grep tty

What to look for:

```
[    0.000000] Kernel command line: console=ttyS0,115200n8 ...
[    1.234567] 44e09000.serial: ttyS0 at MMIO 0x44e09000 (irq = 72, base_baud = 3000000) is a 8250
[    1.234890] 48022000.serial: ttyS1 at MMIO 0x48022000 (irq = 73, base_baud = 3000000) is a 8250
```

Interpretation:

- 44e09000.serial: ttyS0 = UART0 hardware at 0x44E09000 mapped to /dev/ttyS0
- MMIO 0x44e09000 = Memory-mapped I/O address
- irq = 72 = Interrupt number for this UART
- base_baud = 3000000 = Base baud rate (48MHz/16)
- is a 8250 = UART type (8250-compatible)

Why this matters:

- Each ttyS device has a corresponding hardware address
- IRQ numbers are needed for interrupt handling in drivers
- You can trace which physical pins each UART uses

3.3 Find UART Hardware Controllers

ls /sys/bus/platform/devices/ | grep serial

Example Output:

44e09000.serial
serial8250

Interpretation:

- 44e09000.serial = UART0 at 0x44E09000
- serial8250 = Generic 8250 driver handling multiple UARTs

Find all UART addresses:

dmesg | grep "serial:" | grep MMIO

Typical output:

44e09000.serial: ttyS0 at MMIO 0x44e09000
48022000.serial: ttyS1 at MMIO 0x48022000
48024000.serial: ttyS2 at MMIO 0x48024000
481a6000.serial: ttyS3 at MMIO 0x481a6000
481a8000.serial: ttyS4 at MMIO 0x481a8000
481aa000.serial: ttyS5 at MMIO 0x481aa000

3.4 Test UART (Loopback)

Physical loopback: Connect TX pin to RX pin with jumper wire

Terminal 1 - Listen
cat /dev/ttyS1

Terminal 2 - Send
echo "Hello" > /dev/ttyS1

You should see:

- "Hello" appears in Terminal 1
- Confirms UART hardware works

---

4. Verify SPI Interfaces

4.1 Check for SPI Device Files

ls /dev/spi*

Expected if SPI is configured:

/dev/spidev0.0 /dev/spidev0.1 /dev/spidev1.0

If you see:

ls: cannot access '/dev/spi*': No such file or directory

Interpretation:

- SPI hardware exists but is not configured
- Pins are in GPIO mode (not SPI mode)
- No device tree overlay loaded for SPI
- No /dev/spidev driver loaded

4.2 Check SPI in Kernel Messages

dmesg | grep -i spi

If SPI is working, you'd see:

[X.XXXXX] spi spi0.0: setup mode 0, 8 bits/w, 1000000 Hz max --> 0
[X.XXXXX] omap2_mcspi 48030000.spi: SPI Controller at 0x48030000

If nothing appears:

- SPI controllers are not initialized
- Device tree doesn't enable SPI nodes

4.3 Check SPI Hardware Exists

ls /sys/bus/platform/devices/ | grep spi

What you might see:

480ca000.spinlock

Interpretation:

- spinlock is NOT an SPI controller (it's hardware spinlock)
- No SPI platform devices = SPI not enabled in device tree

Why SPI isn't showing:

- Device tree nodes for SPI0/SPI1 are disabled
- Need device tree overlay to enable them

4.4 Check if SPI Kernel Module Exists

lsmod | grep spi

What to look for:

spi_omap2_mcspi   16384 0
spidev   20480 0

If empty:

# Try to load SPI driver manually
sudo modprobe spi_omap2
sudo modprobe spidev

After loading, check again:

ls /dev/spidev*

If still no device:

- Hardware isn't enabled in device tree
- Pins aren't configured for SPI mode

---

5. Understanding Pin Multiplexing

5.1 What is Pinmux?

The AM335x processor has limited pins but many peripherals. Each pin can serve multiple functions:

- **Mode 0-7**: Different peripheral functions
- Example: Pin P9.17 can be:
  - Mode 0: SPI0_CS0
  - Mode 7: GPIO0_5

5.2 View All Pin Configurations

sudo cat /sys/kernel/debug/pinctrl/44e10800.pinmux-pinctrl-single/pins

What it shows:

pin 0 (PIN0) 0:gpio-0-31 44e10800 00000031 pinctrl-single

Breaking down the format:

- pin 0 = Sequential pin number in pinmux controller
- (PIN0) = Kernel pin name
- 0:gpio-0-31 = GPIO chip:pin assignment
- 44e10800 = Pinmux register address for this pin
- 00000031 = Current pinmux value
- pinctrl-single = Driver managing this pin

5.3 Decode Pinmux Values

The pinmux value is a 32-bit register. Lower 8 bits matter most:

Bit 7: Reserved

Bit 6: SLEW (0=fast, 1=slow)
Bit 5: RXACTIVE (0=output, 1=input)
Bit 4: PULLTYPESEL (0=pull-down, 1=pull-up)
Bit 3: PULLUDEN (0=pull enabled, 1=pull disabled)
Bit 2-0: MUXMODE (0-7, selects peripheral function)

Common Values:

- 0x00 = Mode 0, Output, No pull
- 0x20 = Mode 0, Input, No pull
- 0x27 = Mode 7 (GPIO), Pull-down enabled, Input
- 0x30 = Mode 0, Pull-up enabled, Input
- 0x31 = Mode 1, Pull-up enabled, Input
- 0x37 = Mode 7 (GPIO), Pull-up enabled, Input

5.4 Example: Finding SPI0 Pins

sudo cat /sys/kernel/debug/pinctrl/44e10800.pinmux-pinctrl-single/pins | grep -A 1 "44e10950\|44e10954\|44e10958\|44e1095c"

Looking for these registers (SPI0):

- 0x44e10950 = P9.22 (SPI0_SCLK)
- 0x44e10954 = P9.21 (SPI0_D0/MISO)
- 0x44e10958 = P9.18 (SPI0_D1/MOSI)
- 0x44e1095c = P9.17 (SPI0_CS0)

Example Output:

pin 97 (PIN97) 15:gpio-96-127 44e10950 00000037 pinctrl-single
pin 98 (PIN98) 5:gpio-64-95 44e10954 00000037 pinctrl-single
pin 99 (PIN99) 6:gpio-64-95 44e10958 00000037 pinctrl-single
pin 100 (PIN100) 14:gpio-64-95 44e1095c 00000037 pinctrl-single

Interpretation:

- All show 00000037
- Mode = 7 (GPIO mode, not SPI!)
- Pull-up enabled, Input mode
- **Need to change to Mode 0 for SPI function**

5.5 Pin Address to Header Mapping

How to find which physical pin?

1. Check AM335x datasheet "Ball Characteristics" table
2. Use BeagleBone pinout diagrams
3. Cross-reference register offset with header

Example for 0x44e10950:

- Offset from base (0x44e10800): 0x150
- This is pin offset 0x150 / 4 = pin 84 (decimal)
- Maps to ball A17 on processor
- Connects to header **P9.22**

Quick Reference:

Register	Offset	Physical Pin	Default Function

0x44e10800	0x000	P9.1 (GND)	-
0x44e10950	0x150	P9.22	SPI0_SCLK (mode 0)
0x44e10954	0x154	P9.21	SPI0_D0 (mode 0)
0x44e10958	0x158	P9.18	SPI0_D1 (mode 0)
0x44e1095C	0x15C	P9.17	SPI0_CS0 (mode 0)

---

6. Checking Kernel Configuration

6.1 View Kernel Config

```
zcat /proc/config.gz | grep -E "SPI|I2C|SERIAL"
```

What it does:

- Extracts kernel configuration used to build the kernel
- Shows which drivers are built-in (=y) or modules (=m)

6.2 Understanding Config Options

For I2C:

```
CONFIG_I2C=y                # I2C subsystem enabled (built-in)
CONFIG_I2C_CHARDEV=y        # I2C character device support
CONFIG_I2C_OMAP=y           # TI OMAP I2C driver
```

Interpretation:

- =y means compiled into kernel (always available)
- =m means compiled as module (load with modprobe)
- is not set means feature disabled

For SPI:

```
CONFIG_SPI=y                # SPI subsystem enabled
CONFIG_SPI_MASTER=y         # SPI master mode support
CONFIG_SPI_OMAP24XX=y       # TI OMAP2/3/4 SPI driver
CONFIG_SPI_SPIDEV=m         # Userspace SPI driver (module)
```

For UART:

```
CONFIG_SERIAL_8250=y        # 8250/16550 serial driver
CONFIG_SERIAL_8250_OMAP=y   # OMAP 8250 driver
CONFIG_SERIAL_8250_NR_UARTS=6   # Maximum 6 UARTs
```

6.3 Check Loaded Modules

```
lsmod | grep -E "spi|i2c"
```

Example Output:

```
spidev              20480  0
spi_omap2_mcspi       16384  0
i2c_dev               16384  2
```

Interpretation:

- Module is loaded (present in output)
- Number on right = Reference count (how many things use it)

---

7. Finding Pin Mappings

7.1 Physical Pin to Processor Ball

Use BeagleBone System Reference Manual:

- Table 5: P8 Header Pinout
- Table 6: P9 Header Pinout

Example - P9 Header excerpt:

PIN	PROC	MODE0	MODE1	MODE2	MODE3	MODE4	MODE5	MODE6	MODE7
P9.17	A16	SPI0_CS0	MMC2_SDWP	I2C1_SCL	...	...	...	...	gpio0_5
P9.18	B16	SPI0_D1	MMC1_SDWP	I2C1_SDA	...	...	...	...	gpio0_4
P9.21	B17	SPI0_D0	UART2_TXD	I2C2_SCL	...	...	...	...	gpio0_3
P9.22	A17	SPI0_SCLK	UART2_RXD	I2C2_SDA	...	...	...	...	gpio0_2

How to use:

- PIN = Physical header pin
- PROC = Processor ball designation
- MODE0-7 = Different functions this pin can have
- For SPI0, use MODE0

7.2 Processor Ball to Register Address

From AM335x TRM (Chapter 9: Pad Control Module):

- Base address: 0x44E10800
- Each pin has a 4-byte register
- Offset calculated from ball number

Example calculations:

```
conf_spi0_sclk (ball A17) = 0x44E10800 + 0x150 = 0x44E10950
conf_spi0_d0   (ball B17) = 0x44E10800 + 0x154 = 0x44E10954
conf_spi0_d1   (ball B16) = 0x44E10800 + 0x158 = 0x44E10958
conf_spi0_cs0  (ball A16) = 0x44E10800 + 0x15C = 0x44E1095C
```

7.3 Quick Pin Lookup Table

I2C Pins:

Physical Pin	Processor	Function	Register	Mode
P9.19	D17	I2C2_SCL	0x44e1097C	3
P9.20	D18	I2C2_SDA	0x44e10978	3

UART1 Pins:

Physical Pin	Processor	Function	Register	Mode
P9.24	D15	UART1_TXD	0x44e10984	0
P9.26	D16	UART1_RXD	0x44e10980	0

SPI0 Pins:

Physical Pin	Processor	Function	Register	Mode
P9.17	A16	SPI0_CS0	0x44e1095C	0
P9.18	B16	SPI0_D1 (MOSI)	0x44e10958	0
P9.21	B17	SPI0_D0 (MISO)	0x44e10954	0
P9.22	A17	SPI0_SCLK	0x44e10950	0

---

8. Understanding Device Tree

8.1 What is Device Tree?

- Hardware description language
- Tells kernel what hardware exists and how to configure it
- Compiled from `.dts` (source) to `.dtb` (binary)
- Located in `/boot/dtbs/` directory

8.2 Check Active Device Tree

```
# From beagle-version output
sudo beagle-version | grep "Device-Tree"
```

Example:

```
UBOOT: Booted Device-Tree:[am335x-boneblack.dts]
```

This means:

- Base device tree: `am335x-boneblack.dtb`
- Source file was: `am335x-boneblack.dts`

8.3 Examine Device Tree

```
# Decompile device tree to human-readable format
dtc -I fs /sys/firmware/devicetree/base > /tmp/devicetree.dts

# View specific section
```

```
dtc -I fs /sys/firmware/devicetree/base | grep -A 20 "i2c@"
```

Example I2C node:

```
i2c@4819c000 {
    compatible = "ti,omap4-i2c";
    reg = <0x4819c000 0x1000>;
    interrupts = <0x1e>;
    status = "okay";
    clock-frequency = <0x186a0>;  /* 100000 = 100kHz */
    pinctrl-names = "default";
    pinctrl-0 = <&i2c2_pins>;
};
```

Interpretation:

- i2c@4819c000 = I2C2 controller at address 0x4819C000
- compatible = "ti,omap4-i2c" = Driver to use
- status = "okay" = This device is enabled
- clock-frequency = 100000 = 100kHz I2C speed
- pinctrl-0 = <&i2c2_pins> = Use pin configuration named "i2c2_pins"

8.4 Check Device Tree Overlays

```
cat /boot/uEnv.txt | grep overlay
```

What it shows:

- Which device tree overlays are loaded at boot
- Overlays modify the base device tree

Example:

```
enable_uboot_overlays=1
uboot_overlay_addr4=BB-SPI0-01-00A0.dtbo
```

Interpretation:

- Overlays are enabled
- BB-SPI0-01-00A0.dtbo would enable SPI0 if present

8.5 Why SPI Doesn't Work

Check device tree for SPI:

```
dtc -I fs /sys/firmware/devicetree/base | grep -A 10 "spi@"
```

If you see:

```
spi@48030000 {
    compatible = "ti,omap4-mcspi";
    reg = <0x48030000 0x400>;
    status = "disabled";    /* <-- This is why! */
};
```

The problem:

- status = "disabled" means kernel ignores this hardware
- Need device tree overlay to change to status = "okay"
- Without overlay, SPI controller never initializes

---

9. Complete Verification Workflow

Step-by-Step Checklist After Boot:

```
# 1. System Info
uname -r
sudo beagle-version

# 2. Check device files
ls /dev/i2c*
ls /dev/ttyS*
ls /dev/spidev*

# 3. Check kernel messages
dmesg | grep -i i2c
dmesg | grep -i uart
dmesg | grep -i spi

# 4. Check hardware controllers
ls /sys/bus/platform/devices/ | grep -E "spi|i2c|serial"

# 5. Check pin configuration
sudo cat /sys/kernel/debug/pinctrl/44e10800.pinmux-pinctrl-single/pins

# 6. Check kernel config
zcat /proc/config.gz | grep -E "^CONFIG_I2C|^CONFIG_SPI|^CONFIG_SERIAL"

# 7. Check device tree
cat /boot/uEnv.txt
dtc -I fs /sys/firmware/devicetree/base | grep -A 5 "status"
```

---

10. Summary Table

Interface	Device Files	Hardware Detected	Pins Configured	Status
I2C-0	/dev/i2c-0	✅ 44e0b000.i2c	✅ Internal	✅ Working
I2C-2	/dev/i2c-2	✅ 4819c000.i2c	✅ P9.19, P9.20	✅ Working
UART0-5	/dev/ttyS0-5	✅ Multiple	✅ Various pins	✅ Working
SPI0	None	❌ Not in platform	❌ Pins in GPIO mode	❌ Not configured
SPI1	None	❌ Not in platform	❌ Pins in GPIO mode	❌ Not configured

---

11. What's Next?

For I2C Driver Development:

✅ Ready to start - hardware configured

For UART Driver Development:

✅ Ready to start - hardware configured

For SPI Driver Development:

⚠️ Need to enable SPI by either:

1. Creating device tree overlay
2. Configuring pins in your driver
3. Using runtime configuration tools

---

12. Reference Commands Quick Sheet

```
# Check all interfaces at once
ls /dev/{i2c,ttyS,spidev}* 2>/dev/null

# Complete hardware scan
dmesg | grep -iE "i2c|spi|uart|serial|tty"

# Pin configuration
sudo cat /sys/kernel/debug/pinctrl/44e10800.pinmux-pinctrl-single/pins | less

# Kernel modules
lsmod | grep -iE "i2c|spi|serial"

# Device tree info
dtc -I fs /sys/firmware/devicetree/base > /tmp/dt.dts
cat /tmp/dt.dts | grep -E "i2c@|spi@|serial@" -A 10
```

Additional Resources

- **AM335x Technical Reference Manual**: Hardware register details
- **BeagleBone System Reference Manual**: Pin mappings
- **Linux Device Tree Documentation**: /usr/src/linux/Documentation/devicetree/
- **Kernel Source**: /usr/src/linux/ (if installed)

This guide is specifically for BeagleBone Black with kernel 6.12.32-bone28