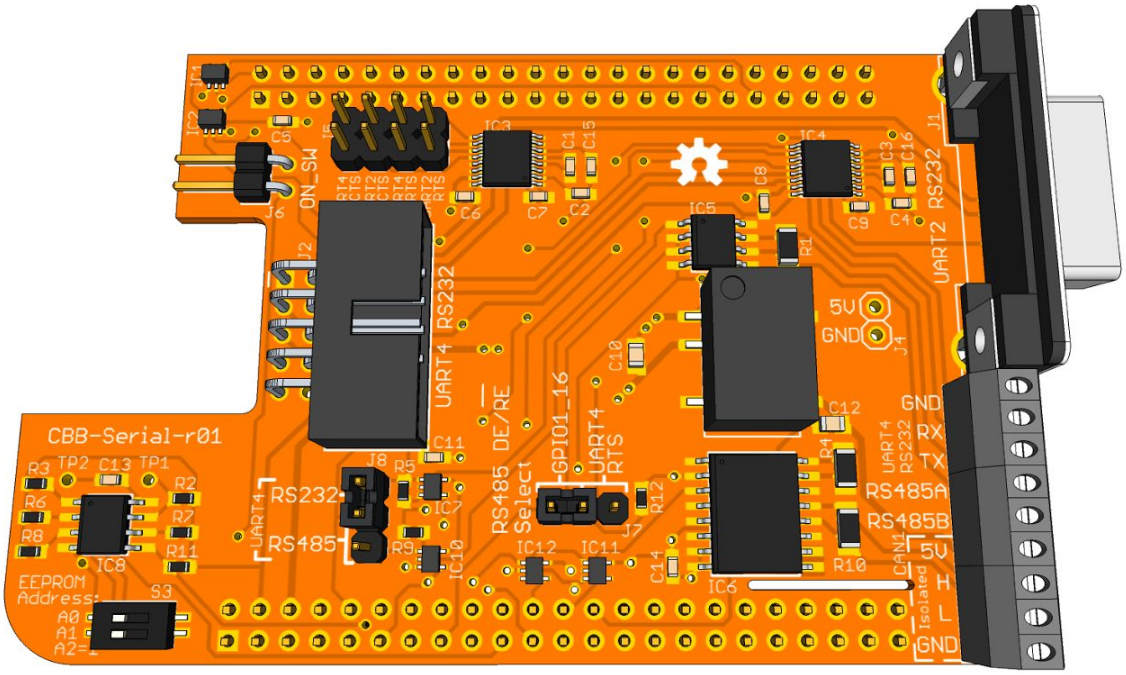
Beagle Bone Black Serial Cape

CBB-Serial

Revision 1.3
Feb 20, 2015



www.logicsupply.com
info@logicsupply.com
+1 802 861 2300

Description

The serial cape was designed for use with both the original BeagleBone and the BeagleBone Black. It allows for the simultaneous use of up to two RS232 ports and one CAN bus, or one RS232 port, one RS485 bus and one CAN bus.

Features

- Two RS232 ports with optional flow control
- Half-duplex RS485 transceiver
- Isolated CAN transceiver with external connections for isolated 5V supply
- Standard IDC10 and DB9 connectors for RS232
- Screw terminals for RS232, RS485 and CAN connections
- 2-pin header for connecting optional external power button
- Standard BeagleBone cape EEPROM with selectable address
- Dimensions: 55 x 87 x 13mm (above P8,P9)
- RoHS compliant

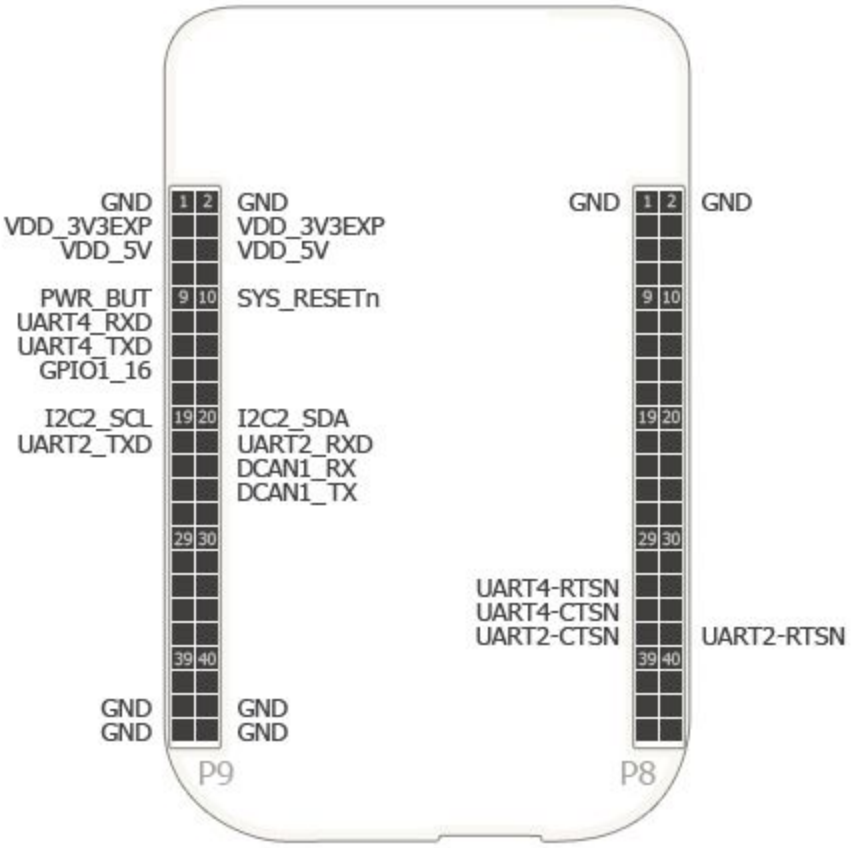
Architecture

Signal name	Header Pin	Pin Mode	Comments
UART4_RTS[1]	P8_33	output	UART4 request-to-send
UART4_CTS[1]	P8_35	input	UART4 clear-to-send
UART2_CTS[1]	P8_37	input	UART4 clear-to-send
UART2_RTS[1]	P8_38	output	UART2 request-to-send
ON_SW	P9_9	input	Trigger power on
UART4_RX[2]	P9_11	RS232 input	±1.6 - ±12V on IDC10 / screw terminal
UART4_TX[2]	P9_13	RS232 output	±5.4V out IDC10 / screw terminal
UART4_RX[2]	P9_11	serial in from RS485	-7 - +12V common-mode voltage on receiver
UART4_TX[2]	P9_13	serial out to RS485	Standard RS485 5V differnetial driver output
GPIO1_6	P9_15	digital output	for software RS485 RE/DE control
UART2_TX	P9_21	RS232 output	±5.4V out IDC10 / screw terminal
UART2_RX	P9_22	RS232 input	±1.6 - ±12V on IDC10 / screw terminal
DCAN1_RX	P9_24	serial in from CAN bus	-27 - +40V common-mode voltage on bus
DCAN1_TX	P9_26	serial out to CAN bus	3V max driver differential output voltage

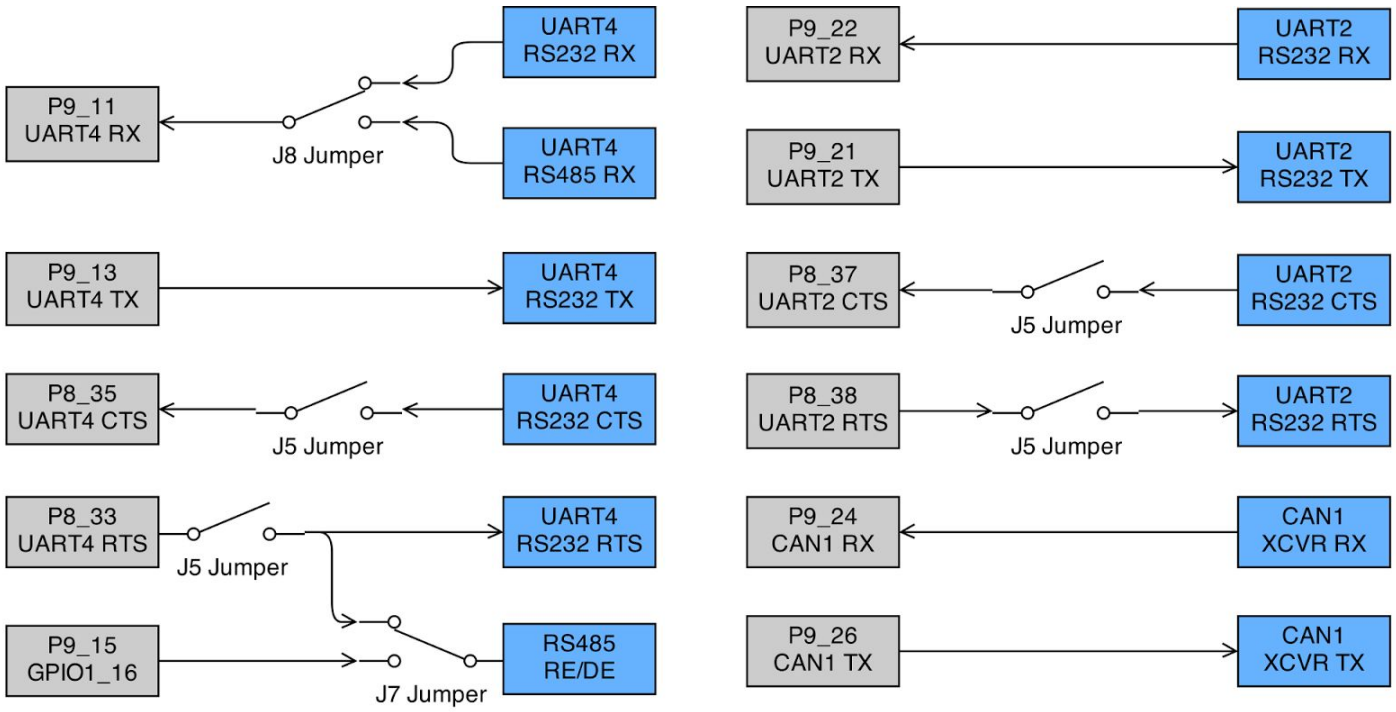
NOTES:

1. These pins are shared with the HDMI driver and are disabled by default. See ‘Enabling Flow Control’ below
2. UART4 is shared between the RS232 converter and the RS485 driver, its function is selected with jumper

BeagleBone Black Pin Allocation



Block Diagram



The BeagleBone Serial cape is designed to work with existing software on the BeagleBone Black.

Requirements for use (not included):

- BeagleBone Black with suitable distribution loaded
- 5V DC Power Supply for BeagleBone Black, 2A recommended

Getting Started

1. Ensure the BeagleBone Black is loaded with a suitable Linux distribution. We recommend using the latest Debian image from http://beagleboard.org/latest-images/, as it will come loaded with the necessary files for the serial cape.
2. If using a different image, download the Device Tree overlay files for capemgr:

```
cd /lib/firmware
wget https://github.com/lgxlogic/CBB-Serial/tree/master/overlay/cape-CBB-Serial-r01.dtbo
wget https://github.com/lgxlogic/CBB-Serial/tree/master/overlay/BB-UART2-RTSCTS-00A0.dtbo
wget https://github.com/lgxlogic/CBB-Serial/tree/master/overlay/BB-UART4-RTSCTS-00A0.dtbo
```

3. Make sure power is disconnected from the BeagleBone Black and connect the Cape by inserting the header pins in headers P8 and P9. Unless you have gone through the steps to disable HDMI, ensure the jumpers are not in place on the flow control enable header (J5).
4. If using in conjunction with other capes, make sure the DIP Switch on every Cape is set to a different EEPROM ID and ensure the different Capes are not using conflicting pins.
5. Connect a 5V DC Power Supply to the BeagleBone Black.
6. The Cape manager will read the eeprom and will configure pins and resources automatically.
7. If you downloaded the Device Tree files manually in step 2, confirm that the overlay loaded successfully:

```
cat /sys/devices/bone_capemgr.*/slots
```

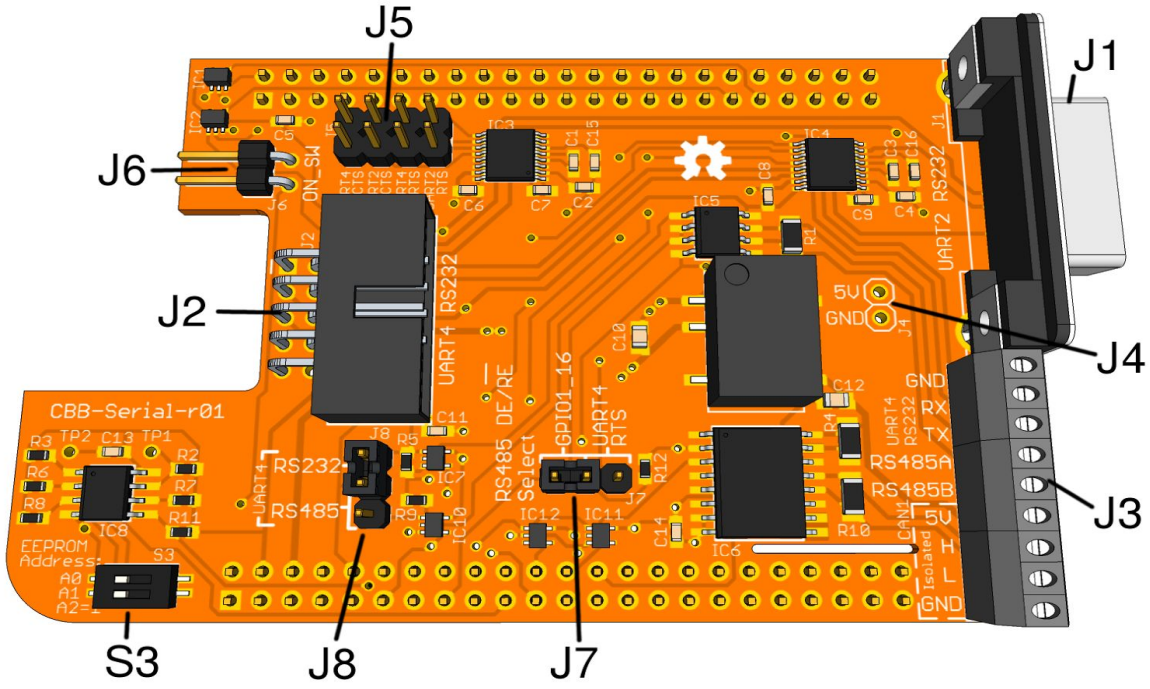
You should see the the line:

```
0: 54:P---L cape-CBB-Serial,r01,Logic Supply,cape-CBB-Serial
```

If 'cape-CBB-Serial' is not in the list you will have to manually load the overlay with:

```
echo cape-CBB-Serial:r01 > /sys/devices/bone_capemgr.*/slots
```

Key Component Locations



J1 - UART2 RS232 DB9 Connector

This connector may be used with a standard 9-pin RS232 cable for communication through UART2. It uses the standard DB9 pinout for DTE (data terminal equipment - same as on a PC motherboard):

J1	Name	Description
Pin 1	DCD	Not used on serial cape
Pin 2	DSR	Not used on serial cape
Pin 3	RXD	Connected to UART2 RS232 RX (receive)
Pin 4	RTS	Connected to UART2 RS232 RTS (request-to-send)
Pin 5	TXD	Connected to UART2 RS232 TX (transmit)
Pin 6	CTS	Connected to UART2 RS232 CTS (clear-to-send)
Pin 7	DTR	Not used on serial cape
Pin 8	RI	Not used on serial cape
Pin 9	GND	Connected to BeagleBone GND

J2 - UART4 RS232 IDC10 Header

This header is for connecting an IDC10 -> DB9 RS232 cable for RS232 communication through UART4. It uses the standard DTK/Intel pinout:

J2	Name	Description
Pin 1	DCD	Not used on serial cape
Pin 2	DSR	Not used on serial cape
Pin 3	RXD	Connected to UART4 RS232 RX (receive)
Pin 4	RTS	Connected to UART4 RS232 RTS (request-to-send)
Pin 5	TXD	Connected to UART4 RS232 TX (transmit)
Pin 6	CTS	Connected to UART4 RS232 CTS (clear-to-send)
Pin 7	DTR	Not used on serial cape
Pin 8	RI	Not used on serial cape
Pin 9	GND	Connected to BeagleBone GND
Pin 10	N.C.	Unused pin (removed on J2)

J3 - RS232/RS485/CAN Screw Terminal Connections

This connector provides access to the RS485 and CAN transceivers, also be provides an alternative connection to UART4.

J3	Name	Description
Pin 1	CAN GND	GND for the isolated CAN supply
Pin 2	CAN L	CAN low (CAN-) - inverted signal
Pin 3	CAN H	CAN high (CAN+) - non-inverted signal
Pin 4	CAN 5V	Isolated supply for CAN transceiver - 5V 100mA
Pin 5	RS485 A	(RS485-) - inverted signal
Pin 6	RS485 B	(RS485+) - non-inverted signal
Pin 7	U4 TX	Connected to UART4 RS232 TX (transmit)
Pin 8	U4 RX	Connected to UART4 RS232 RX (receive)
Pin 9	GND	Connected to BeagleBone GND

J4 - 5V Supply

These pins can be used to power external devices directly from the DC barrel jack input on the BeagleBone.

J5 - RS232 Flow Control Enable

This connector is used to enable flow control on UART2 and UART4. See the **Enabling Flow Control** section below.

J5	Name	Description
Pos 1	UART4 CTS	Place a jumper here to connect through the UART4 CTS signal
Pos 2	UART2 CTS	Place a jumper here to connect through the UART2 CTS signal
Pos 3	UART4 RTS	Place a jumper here to connect through the UART4 RTS signal
Pos 4	UART2 RTS	Place a jumper here to connect through the UART2 RTS signal

J6 - ON_SW

Used to turn on the BeagleBone. Connecting these two pins will trigger the boot sequence if the BeagleBone is powered off (e.g. with the command `# sudo shutdown -h now`).

J7 -RS485 DE/RE Control Select

This header is used to select whether the RS485 transceiver is toggled between transmit and receive modes by the user with GPIO1_16, or automatically by the kernel with UART4 RTS. See the **RS485** section below.

J7	Name	Description
Pos 1	GPIO1_16	Place a jumper here to connect GPIO1_16 to the RS485 transceivers DE/RE pins
Pos 2	UART4 RTS	Place a jumper here to connect UART4 RTS to the RS485 transceivers DE/RE pins

J8 -UART4 Function Select

This header is used to select whether UART4 will receive RS232 or RS485 data. See the **RS485** section below.

J8	Name	Description
Pos 1	RS232	Place a jumper here if using UART4 for RS232 communication
Pos 2	UART4 RTS	Place a jumper here if using UART4 for RS485 communication

S3 -EEPROM Address

This switch sets the I2C address of the cape EEPROM. If using the serial cape in conjunction with other capes, ensure that all are set to have different addresses prior to booting.

EEPROM Details

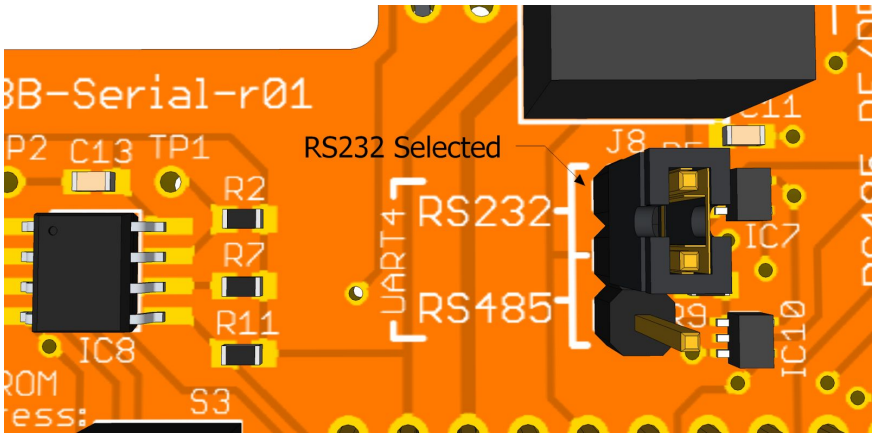
The EEPROM on the Cape is used to load the BeagleBone Black with the appropriate configuration for the Cape.

EEPROM Support: YES
Board Name: CBB-Serial
Manufacturer: Logic Supply

Programming

RS232

First, if using UART4, ensure that the J8 jumper is in the RS232 position, which connects the RS232 RX to the BeagleBone:



If using UART4, either connect an IDC10-RS232 cable to J2 (DTK/Intel pinout), or connect RX and TX lines to the screw terminal. If using UART2 connect a serial cable to the J1 DB9 connector. When running Linux, the BeagleBone's serial ports are set up as DTEs (Data Terminal Equipment), so a crossover RS232 cable will be required if connecting to a PC.

Using the RS232 ports without flow control requires no additional software setup. When the BeagleBone boots up with the serial cape attached, the kernel will create the special files /dev/ttyO2 for UART2 and /dev/ttyO4 for UART4. Simply pass these files to whatever serial tool or library you are using. For example, with Python's pyserial library:

```
import time, serial
uart4_file = '/dev/ttyO4'
baud = 115200
ser = serial.Serial(uart4_file, baud)
while True:
  ser.write("Testing")
  time.sleep(1)
```

Will write the string 'Testing' out of UART4 at a baud rate of 115200 once every second until you press ctrl-c. You can install PySerial on Angstrom with:

```
opkg update
opkg install python-serial
```

And on Debian with:

```
apt-get update
apt-get install python-serial
```

Enabling Flow Control

Because the RTS and CTS pins are shared with the HDMI driver, to enable flow control for either UART4 or UART2 you must first disable HDMI. To do so, first log in to your BegleBone's root account, either by using SSH to log in directly, or by running `# sudo -s` from a normal user account. Next you need to mount the boot partition and open the boot configuration file:

```
mkdir /mnt/boot
mount /dev/mmcblk0p1 /mnt/boot/
nano /mnt/boot/uEnv.txt
```

Add the following line to the file:

```
optargs=capemgr.disable_partno=BB-BONELT-HDMI,BB-BONELT-HDMIN
```

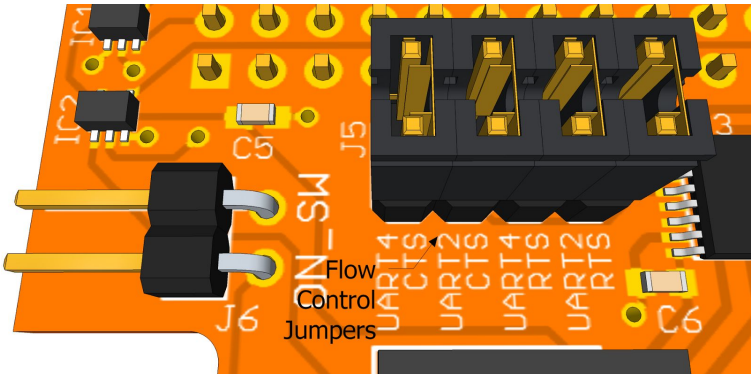
Save and close the file by pressing `<ctrl>-X, Y, <enter>`, then run `# reboot` to reboot your BeagleBone. Once the BeagleBone boots back up, confirm that HDMI has been successfully disabled with

```
cat /sys/devices/bone_capemgr.*/slots
```

You should see the lines:

```
4: ff:P-O-L Bone-LT-eMMC-2G,00A0,Texas Instrument,BB-BONE-EMMC-2G
5: ff:P-O-- Bone-Black-HDMI,00A0,Texas Instrument,BB-BONELT-HDMI
6: ff:P-O-- Bone-Black-HDMIN,00A0,Texas Instrument,BB-BONELT-HDMIN
```

Note that there's no 'L' on the left of the last two rows; this means that those two overlays are not enabled. Once HDMI is disabled you can place the four jumpers on the flow control enable header (J5):



HDMI should be disabled before placing the headers, as having them in place while HDMI is enabled can interfere with the boot process.

Next you'll need to apply the appropriate flow control Device Tree overlay. For UART2:

```
echo BB-UART2-RTSCTS > /sys/devices/bone_capemgr.*/slots
```

And for UART4:

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

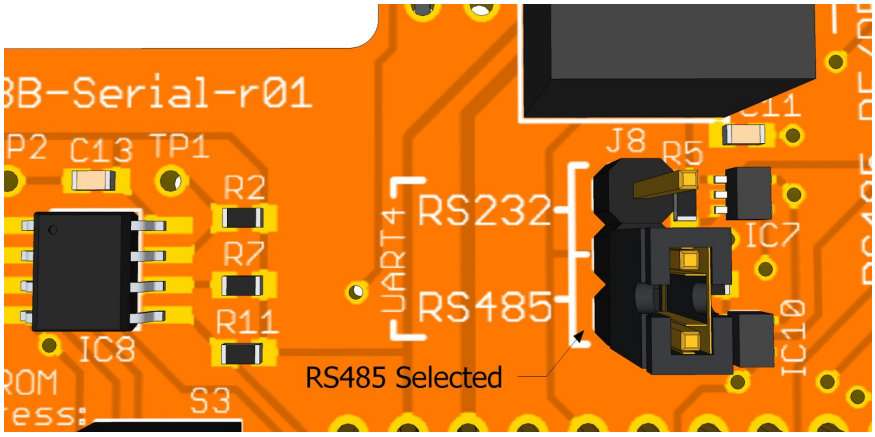
The UART can now use RTS/CTS handshaking. For example in Python:

```
import time, serial
uart4_file = '/dev/ttyO4'
baud = 115200
ser = serial.Serial(uart4_file, baud, rtscts=True)
while True:
  ser.write("Testing")
  time.sleep(1)
```

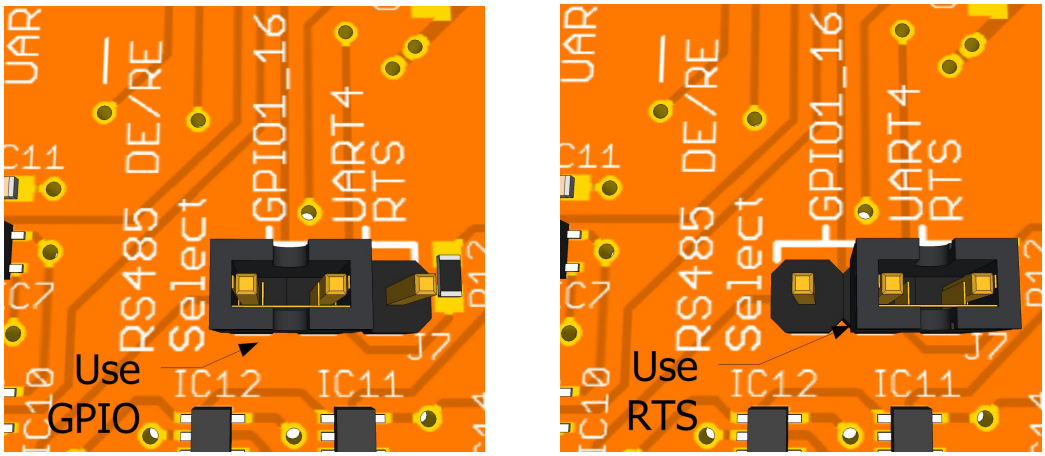
Note: If you are using UART4 for RS232 while you have the RS485 transceiver connected to a bus, you must place the J7 jumper into the GPIO1_16 position and set GPIO1_16 low to prevent data from being transmitted on the RS485 bus.

RS485

UART4 is used for RS485 communication, so UART4 RS232 and RS485 can not be used simultaneously. To use RS485, first ensure the J8 jumper is in the RS485 position to allow received data to be passed to the UART:



Use the J7 jumper to set which pin is used to toggle the transceiver between transmit and receive modes:



If you plan to use GPIO1_16 to manually toggle between driving and receiving on the bus, first export it from Kernel control to user control and set it as an output:

```
echo 48 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio48/direction
```

You can then put the RS485 transceiver into transmit mode by writing '1' to the file '/sys/class/gpio/gpio48/value', and put it in receive mode by writing '0' to the same file. Because the GPIO pin is being controlled through the filesystem from userspace there will be additional latency.

Alternatively, the Linux Kernel can handle switching between transmit and receive modes using UART4's RTS output. To take advantage of this, first follow the steps above under 'Enabling Flow Control'. You then just need to put UART4 into RS485 mode using ioctl. In Python this is as simple as:

```
import time, struct, fcntl, serial
uart4_file = '/dev/ttyO4'
baud = 115200
ser = serial.Serial(uart4_file, baud)
fd=ser.fileno()
serial_rs485 = struct.pack('hhhhhhhh', 1, 0, 0, 0, 0, 0, 0, 0)
fcntl.ioctl(fd,0x542F,serial_rs485)
while True:
  ser.write("Testing")
  time.sleep(1)
```

Unfortunately, not all BeagleBone kernel builds have RS485 support enabled, in which case running the above program will throw this error:

IOError: [Errno 25] Inappropriate ioctl for device

If this is the case you can try updating your Angstrom system with:

```
opkg update
opkg upgrade
```

Or Debian with:

```
apt-get update
apt-get upgrade
apt-get dist-upgrade
```

If it still fails you can manually control the RTS line until the kernel gets patched for RS485 support.

CAN Bus

Your BeagleBone should already have the SocketCAN drivers installed, so to use the CAN bus you just need to set the bitrate and bring up the CAN0 interface:

```
sudo ip link set can0 up type can bitrate 125000
```

Note that the pins on the serial cape are marked as CAN1, but we used CAN0 here. The AM335x microprocessor's CAN modules are numbered starting at 1, and the Linux kernel numbers the CAN interfaces starting at 0. The serial cape is labeled to match the AM335x numbering scheme to be consistent with the datasheet and schematic symbols.

To test the CAN interface from the command line you'll want the can-utils package. You can run `# which candump` to test if you have it installed. If that prints nothing then you'll need to download and compile can-utils:

```
opkg install git
```

Or on Ubuntu/Debian:

```
apt-get install git
```

Then:

```
cd ~
git clone git://gitorious.org/linux-can/can-utils.git
cd can-utils
make
make install
```

Then to monitor the CAN bus run:

```
candump can0
```

If your BeagleBone is connected to an active CAN bus you should begin to see data in the terminal.

If you would like to test the CAN setup without connecting to an active CAN bus this can be accomplished with 2 Beaglebone Blacks and 2 CBB-Serial capes by connecting their Gnd, CAN-H, and CAN-L signals together respectively. After bringing both CAN Interfaces up you can open two terminal windows and run `candump can0` in one window (first) and then use tools such as `cansend` and `cangen` to generate data on the CAN Bus. Because local loopback is on by default you will see whatever packets you send show up in the local *(i.e. same Beaglebone as cansend runs on)* `candump` output if you run it on the same system from which the data is sent as well as on the second system .

Important:
You cannot just test the CAN Bus software without connecting the CBB-Serial hardware to another CAN device. You need to be connected to a receiving CAN device so that ACK (acknowledge) packets get sent and the CAN sender can mark it as sent (i.e. stop buffering it), otherwise the buffer gets full after a short period of time and all future writes to the CAN Bus will fail.

Note:
You can also use the isolated CAN supply to power external devices. It can supply 5V at up to 100mA. Exceeding 100mA will cause the voltage to drop, and will affect the CAN transceiver.

Open Source Hardware

This product is Open Source Hardware. Design materials, schematics and source code is available on GitHub at https://github.com/lgxlogic/CBB-Serial

Design materials are NOT SUPPORTED and DO NOT constitute a reference design. THERE IS NO WARRANTY FOR THE DESIGN MATERIALS, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE DESIGN MATERIALS “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE DESIGN MATERIALS IS WITH YOU. SHOULD THE DESIGN MATERIALS PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
We mean it, these design materials may be totally unsuitable for any purposes.

License

Change History

01/22/2014	Version 1.0	Initial draft Preliminary Release
05/12/2014	Version 1.1	Added Debian instructions
02/19/2015	Version 1.2	Changed UART to BB-UART
02/20/2015	Version 1.3	Removed # from CLI examples to facilitate easy bulk copy and paste operations Removed outdated link to image updating instructions from circuitco Added notes about basic CAN Bus setup testing with just the Beagleboard

More Information

For more information, see www.logicsupply.com

Google+

Follow us on Google+ for product updates.