# UART Communication Beaglebone Black in C

For Starting First you need to check the all Available Uart in BBB for that Use

“ **ls -l /dev/ttyO\*** ”

In BBB the UART PINS are

UART0 (console header) /dev/ttyS0 -- note: /sbin/agetty is running on this port  
UART1 Tx - P9.24, Rx - P9.26 /dev/ttyS1   
UART2 Tx - P9.21, Rx - P9.22 /dev/ttyS2  
UART3 Tx - P9.42 (Tx only) /dev/ttyS3  
UART4 Tx - P9.13, Rx - P9.11 /dev/ttyS4  
UART5 Tx - P8.37, Rx - P8.38 /dev/ttyS5

You need to enable the pins with uBoot overlays:

You need to edit /boot/uEnv.txt:

Change these two lines: (Change as per Uart that you are used)

#uboot\_overlay\_addr6=/lib/firmware/<file6>.dtbo  
#uboot\_overlay\_addr7=/lib/firmware/<file7>.dtbo

to

uboot\_overlay\_addr6=/lib/firmware/BB-UART1-00A0.dtbo  
uboot\_overlay\_addr7=/lib/firmware/BB-UART2-00A0.dtbo

and then reboot.

Then Refer :

[Linux Serial Ports Using C/C++ | mbedded.ninja](https://blog.mbedded.ninja/programming/operating-systems/linux/linux-serial-ports-using-c-cpp/)

For Detail Information (the Code given in Site is Fully Working).

Aslo Refer :

[Serial Programming/termios - Wikibooks, open books for an open world](https://en.wikibooks.org/wiki/Serial_Programming/termios)

In Below Example we use UART 2 (Tested Code in BBB) debian OS. Work with Linux.

**Example for UART Write :**

// C library headers

#include <stdio.h>

#include <string.h>

// Linux headers

#include <fcntl.h> // Contains file controls like O\_RDWR

#include <errno.h> // Error integer and strerror() function

#include <termios.h> // Contains POSIX terminal control definitions

#include <unistd.h> // write(), read(), close()

int main() {

// Open the serial port. Change device path as needed (currently set to an standard FTDI USB-UART cable type device)

int serial\_port = open("/dev/ttyO2", O\_RDWR);

// Create new termios struct, we call it 'tty' for convention

struct termios tty;

// Read in existing settings, and handle any error

if(tcgetattr(serial\_port, &tty) != 0) {

printf("Error %i from tcgetattr: %s\n", errno, strerror(errno));

return 1;

}

tty.c\_cflag &= ~PARENB; // Clear parity bit, disabling parity (most common)

tty.c\_cflag &= ~CSTOPB; // Clear stop field, only one stop bit used in communication (most common)

tty.c\_cflag &= ~CSIZE; // Clear all bits that set the data size

tty.c\_cflag |= CS8; // 8 bits per byte (most common)

tty.c\_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware flow control (most common)

tty.c\_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)

tty.c\_lflag &= ~ICANON;

tty.c\_lflag &= ~ECHO; // Disable echo

tty.c\_lflag &= ~ECHOE; // Disable erasure

tty.c\_lflag &= ~ECHONL; // Disable new-line echo

tty.c\_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP

tty.c\_iflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow ctrl

tty.c\_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL); // Disable any special handling of received bytes

tty.c\_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g. newline chars)

tty.c\_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed

// tty.c\_oflag &= ~OXTABS; // Prevent conversion of tabs to spaces (NOT PRESENT ON LINUX)

// tty.c\_oflag &= ~ONOEOT; // Prevent removal of C-d chars (0x004) in output (NOT PRESENT ON LINUX)

tty.c\_cc[VTIME] = 100; // Wait for up to 1s (10 deciseconds), returning as soon as any data is received.

tty.c\_cc[VMIN] = 0;

// Set in/out baud rate to be 9600

cfsetispeed(&tty, B9600);

cfsetospeed(&tty, B9600);

// Save tty settings, also checking for error

if (tcsetattr(serial\_port, TCSANOW, &tty) != 0) {

printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));

return 1;

}

// Prompt user for input

char user\_input[256];

printf("Enter a string to send over UART: ");

fgets(user\_input, sizeof(user\_input), stdin);

// Add \r\n at the end of the user input

strcat(user\_input, "\r\n");

// Send the user input over UART

write(serial\_port, user\_input, strlen(user\_input));

close(serial\_port);

return 0; // success

};

**Example for UART Read :**

// C library headers

#include <stdio.h>

#include <string.h>

// Linux headers

#include <fcntl.h> // Contains file controls like O\_RDWR

#include <errno.h> // Error integer and strerror() function

#include <termios.h> // Contains POSIX terminal control definitions

#include <unistd.h> // write(), read(), close()

#define MAX\_BUFFER\_SIZE 256

int main() {

// Open the serial port. Change device path as needed (currently set to an standard FTDI USB-UART cable type device)

int serial\_port = open("/dev/ttyO2", O\_RDWR);

// Create new termios struct, we call it 'tty' for convention

struct termios tty;

// Read in existing settings, and handle any error

if(tcgetattr(serial\_port, &tty) != 0) {

printf("Error %i from tcgetattr: %s\n", errno, strerror(errno));

return 1;

}

tty.c\_cflag &= ~PARENB; // Clear parity bit, disabling parity (most common)

tty.c\_cflag &= ~CSTOPB; // Clear stop field, only one stop bit used in communication (most common)

tty.c\_cflag &= ~CSIZE; // Clear all bits that set the data size

tty.c\_cflag |= CS8; // 8 bits per byte (most common)

tty.c\_cflag &= ~CRTSCTS; // Disable RTS/CTS hardware flow control (most common)

tty.c\_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)

tty.c\_lflag &= ~ICANON;

tty.c\_lflag &= ~ECHO; // Disable echo

tty.c\_lflag &= ~ECHOE; // Disable erasure

tty.c\_lflag &= ~ECHONL; // Disable new-line echo

tty.c\_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP

tty.c\_iflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow ctrl

tty.c\_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL); // Disable any special handling of received bytes

tty.c\_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g. newline chars)

tty.c\_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed

// tty.c\_oflag &= ~OXTABS; // Prevent conversion of tabs to spaces (NOT PRESENT ON LINUX)

// tty.c\_oflag &= ~ONOEOT; // Prevent removal of C-d chars (0x004) in output (NOT PRESENT ON LINUX)

tty.c\_cc[VTIME] = 10; // Wait for up to 1s (10 deciseconds), returning as soon as any data is received.

tty.c\_cc[VMIN] = 0;

// Set in/out baud rate to be 9600

cfsetispeed(&tty, B9600);

cfsetospeed(&tty, B9600);

// Save tty settings, also checking for error

if (tcsetattr(serial\_port, TCSANOW, &tty) != 0) {

printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));

return 1;

}

// Receive and print messages

char rx\_buffer[MAX\_BUFFER\_SIZE];

char rx\_temp\_buffer[MAX\_BUFFER\_SIZE];

int rx\_temp\_length = 0;

while (1) {

int rx\_length = read(serial\_port, &rx\_temp\_buffer, sizeof(rx\_temp\_buffer));

if (rx\_length < 0) {

perror("Error reading from UART");

break;

} else if (rx\_length == 0) {

// No data available

usleep(10000); // Small delay to prevent busy-waiting

} else {

// Copy received data to the buffer

memcpy(rx\_buffer + rx\_temp\_length, rx\_temp\_buffer, rx\_length);

rx\_temp\_length += rx\_length;

// Check if a newline character is received

if (strchr(rx\_buffer, '\n') != NULL) {

// Null-terminate the received string

rx\_buffer[rx\_temp\_length] = '\0';

// Print the received string

printf("Received: %s", rx\_buffer);

// Reset the temporary buffer

memset(rx\_buffer, 0, sizeof(rx\_buffer));

rx\_temp\_length = 0;

}

}

}

close(serial\_port);

return 0; // success

};