Serial Communication Between CyberBrick Receiver and Arduino

1. Objective

This guide explains how to make the CyberBrick receiver (based on an ESP32-C3 core board) read commands from the paired remote control and forward them via serial (UART) to an Arduino Mega (or any other Arduino board), which will use them to control its own actuators (motors, servos, etc.). This is particularly useful for making an existing project remotely controllable. In my case, for example, I integrated this system to control my 3D printed lawnmower with a remote controller.

The connection is intentionally one-way: the CyberBrick sends data and the Arduino receives it. No commands are sent from the Arduino back to the CyberBrick.

2. Required Hardware

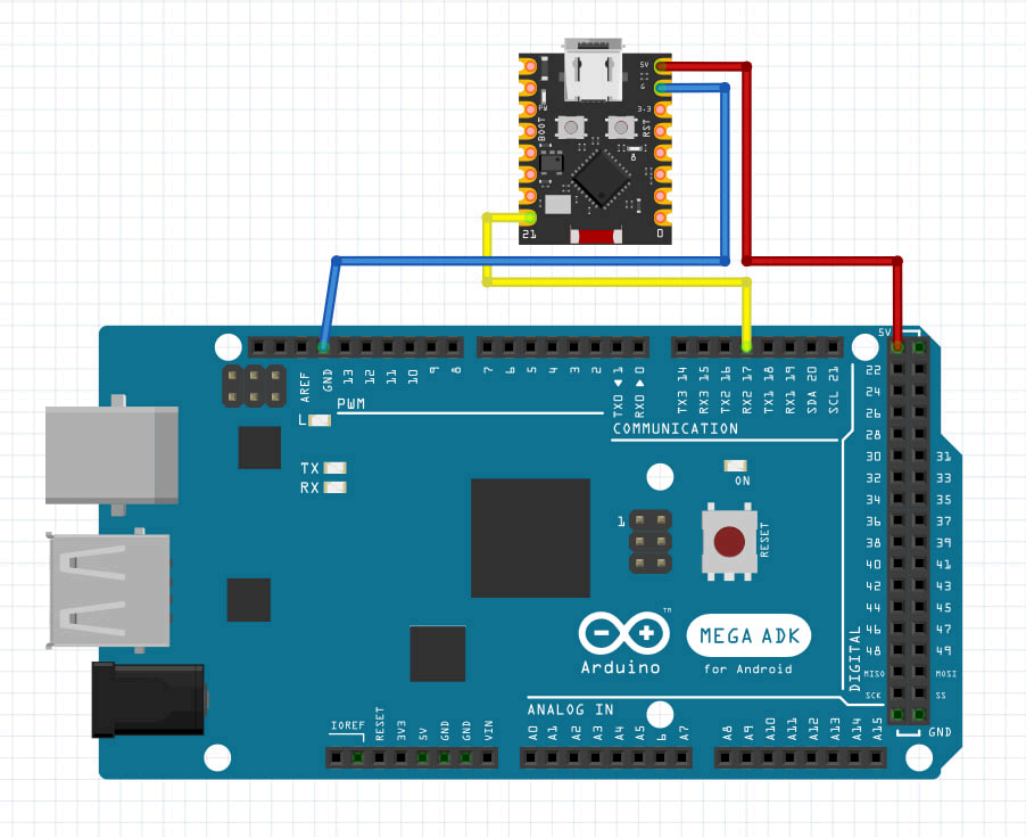
- CyberBrick receiver with ESP32-C3 core board
- CyberBrick remote control kit (transmitter)
- Arduino (I used Mega, but it should work with any other Arduino board.
- Jumper wires
- PC with Thonny (or Arduino Lab for MicroPython) to program the core board, and Arduino IDE for the Mega

3. Physical Connection

Only one signal wire is required because the communication is one-way.

Please note: you will see in the following instructions the use of Serial2 on the Arduino Mega. I simply used Serial2 instead of Serial1 because Serial1 was already in use for different purposes. If you use Arduino UNO, Serial2 is not available. You can simply change the lines of code with Serial1 instead of Serial2, and use the right input pin on the Arduino (search for GPIO schema to know which is the correct one).

CyberBrick (receiver)	Arduino Mega
GPIO21 — TX (pin "T0" on the core board)	Pin 17 — RX2 (Serial2)
VBAT	5V
GND	GND



Note: the CyberBrick RX pin (GPIO20 / D2) should not be connected to anything; it is not needed because the Mega does not send commands back.

Note: the CyberBrick TX signal is 3.3 V, and the Mega correctly reads it as HIGH on its 5 V input (the input threshold is approximately 0.6–0.7×VCC); no resistor divider or level shifter is required in this direction.

Note: the Mega has 4 hardware serial ports (Serial, Serial1, Serial2, Serial3). If pin 17 is occupied, another one can be used (e.g. Serial1 on pin 19, Serial3 on pin 15), with the code updated accordingly.

4. CyberBrick Side — Reading Remote-Control Data

4.1 Module API

Functions used, verified with the command dir(bbl_rc_slave) on the board:

- init() — initializes the RC receiver module
- rc_is_connected() — returns True/False depending on whether the transmitter is paired and connected
- rc_data() — returns a tuple of 10 values: (L1, L2, L3, R1, R2, R3, K1, K2, K3, K4)

The first 6 values are ADC readings in the 0–4095 range (analog axes of the two joysticks and the 3-position selector); the last 4 are digital button states, 0/1.

4.2 Why UART(1) Instead of UART(0)

Instantiating UART(0) on GPIO20/21 generates the error OSError: (-259, 'ESP_ERR_INVALID_STATE') because that peripheral is already occupied by the system. The verified solution is to use UART(1): thanks to the ESP32-C3 GPIO matrix, it can still be routed to the same physical pins, GPIO21 (TX) / GPIO20 (RX), without conflicts.

4.3 Data Transmission Script (Standalone Example)

For debugging and software upload on the board, I used **Arduino Lab for MicroPython** software.

Minimal version, intended to be uploaded and run on its own (useful for a quick initial test before integrating it into the complete RC firmware):

```
from machine import UART, Pin
import time
import bbl_rc_slave

bbl_rc_slave.init()

BAUDRATE = 115200
uart = UART(1, baudrate=BAUDRATE, tx=Pin(21), rx=Pin(20))

SEND_INTERVAL_MS = 20

def format_data(data):
    """data = (L1,L2,L3,R1,R2,R3,K1,K2,K3,K4) -> CSV + newline"""
    return ",".join(str(v) for v in data) + "\n"

def main():
    while True:
        if bbl_rc_slave.rc_is_connected():
            data = bbl_rc_slave.rc_data()
            if data:
                print("WRITING DATA: ", data)
                uart.write(format_data(data))
        time.sleep_ms(SEND_INTERVAL_MS)

if __name__ == "__main__":
    main()
```

- Connect your receiver board via USB to your laptop, open the software and click "Connect" in the top left corner.
- Once connected, copy and paste the previous code in the editor and run it.
- Turn on your remote, (pair it with the receiver by following the official CyberBrick guide if you don't know how to do it), and try to move your joysticks. You will see some Serial print on the debug console, that will represent the commands in the format `[L1,L2,L3,R1,R2,R3,K1,K2,K3,K4].` These strings are exactly what we will receive on the Arduino Serial.

4.4 Integration into the Official RC Firmware (rc_main.py)

So far, we've only run the debug code on our receiver. To upload it to the board and ensure it runs every time we power it on, we need to insert it into the file called *rc_main.py.*

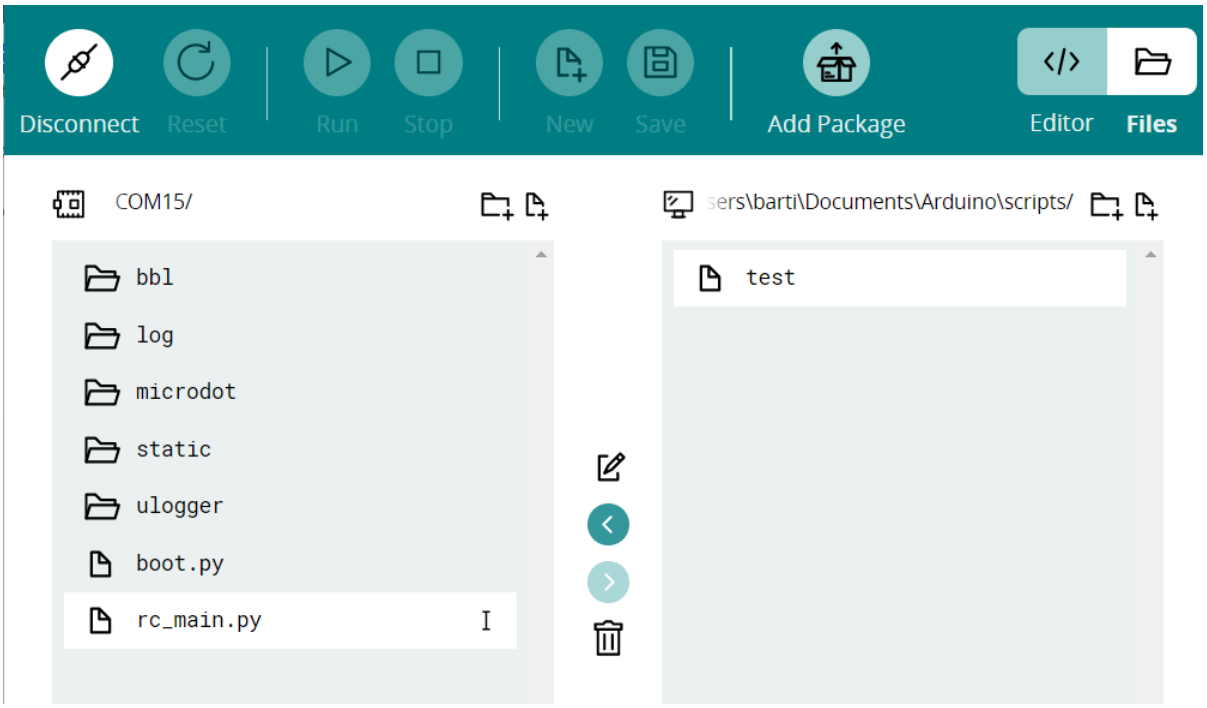
We have two options:

1. Replace the entire contents of *rc_main.py* with the new code that only writes commands to the serial port (this is sufficient for our purposes).

2. Integrate the code into the existing *rc_main.py* to keep the original functionalities.

I chose option one because I wasn't interested in maintaining the receiver's original functionality, and I recommend you to do the same. You can always go back with the original one. Follow these steps:

- **Before proceeding, go to the "Files" section of Arduino Lab and find the *rc_main.py* file. Save the contents of the file somewhere so you can quickly restore the original functionality if needed.**



Now we are ready to upload the new code in *rc_main.py*. Double click on *rc_main.py* and once the editor shows you the content, select all, delete and paste the micro python in 4.3 section. By saving, the file will be automatically updated on the board too.

- Click on the Reset button on the top of Arduino Lab. The board should now run the new code. Try again with the remote controller to see if the serial prints are still prompted while moving your joysticks.
- To find out which input corresponds to which value in the array (L1, L2, etc.), we need to run some tests. Move just one command at a time and see which value changes within the array. Take note of the values corresponding to the various buttons and joysticks; they'll come in handy when you implement the logic on the Arduino.
- (optional) remove the “print” from the micro python and save again.

5. Arduino Mega Side — Receiving and Interpreting Data

On the Mega, the connected hardware serial port (Serial2 in the example) is read, one line is accumulated up to the \n terminator, and it is then parsed using commas as separators. The following code is a starting point to read the commands, but the logic must be implemented according to your needs. You can upload this code to test if commands arrive from the receiver:

```
const uint8_t NUM_VALUES = 10;
int rcValues[NUM_VALUES]; // L1,L2,L3,R1,R2,R3,K1,K2,K3,K4

const unsigned long BAUDRATE = 115200;
```

```
const unsigned long RC_TIMEOUT_MS = 500;
unsigned long lastRxMillis = 0;

void setup() {
  Serial.begin(115200);   // USB, per debug
  Serial2.begin(BAUDRATE); // collegata al CyberBrick (RX2 = pin 17)
}

void loop() {
  if (Serial2.available()) {
    String line = Serial2.readStringUntil('\n');
    if (parseLine(line)) {
      lastRxMillis = millis();
      onNewRcData();
    }
  }

  // Failsafe
  if (millis() - lastRxMillis > RC_TIMEOUT_MS) {
    // es. driveMotors(0, 0);
  }
}

bool parseLine(const String &line) {
  int idx = 0, start = 0;
  for (int i = 0; i <= (int)line.length(); i++) {
    if (i == (int)line.length() || line[i] == ',') {
      if (idx >= NUM_VALUES) return false;
      String token = line.substring(start, i);
      token.trim();
      if (token.length() == 0) return false;
      rcValues[idx] = token.toInt();
      idx++;
      start = i + 1;
    }
  }
  return idx == NUM_VALUES;
}

void onNewRcData() {
  // rcValues[0..9] = L1,L2,L3,R1,R2,R3,K1,K2,K3,K4

  int L1 = rcValues[0];
  int L2 = rcValues[1];
  int L3 = rcValues[2];
  int R1 = rcValues[3];
  int R2 = rcValues[4];
  int R3 = rcValues[5];
  int K1 = rcValues[6];
  int K2 = rcValues[7];
  int K3 = rcValues[8];
  int K4 = rcValues[9];

  Serial.print("L1="); Serial.print(L1);
  Serial.print(" L2="); Serial.print(L2);
  Serial.print(" L3="); Serial.print(L3);
  Serial.print(" R1="); Serial.print(R1);
  Serial.print(" R2="); Serial.print(R2);
  Serial.print(" R3="); Serial.print(R3);
  Serial.print(" K1="); Serial.print(K1);
  Serial.print(" K2="); Serial.print(K2);
  Serial.print(" K3="); Serial.print(K3);
  Serial.print(" K4="); Serial.println(K4);
```

```
}
```

Note: the failsafe is important: without it, if the signal is lost while a command is in progress (e.g. “move forward”), the last command would remain active indefinitely. RC_TIMEOUT_MS defines how many milliseconds without packets are required before the connection is considered lost.

6. Final test

Turn ON your Arduino. Your receiver board will be powered from the Arduino if you followed the previous schema.
Turn ON your remote controller and open the Serial monitor on Arduino IDE.
By moving your joysticks, you should see the commands on the Serial monitor.
You can now implement your logic to transform the commands received in real functionalities!

Remote controller by BambBam design
3D printed LawnMover project from repalmakershop.com