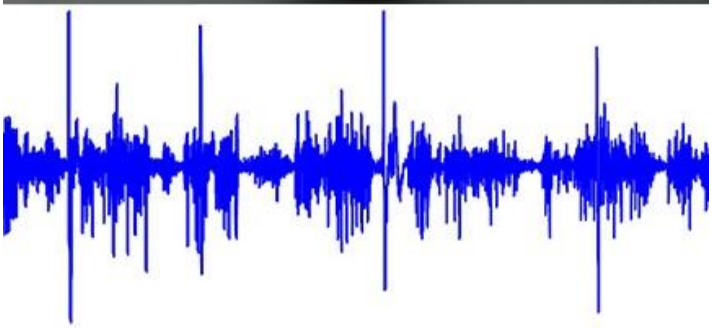


EMG CONTROLLED ROBOT HAND USER MANUAL

A Research Seminar Project from WiSe 2022/2023

HAW Hamburg – Master Biomedical Engineering

SUMMARY

Instructions for setting up an EMG controlled robot hand using Simulink, BITalino, and Arduino. The setup of the serial communication between the components and interchangeable control parameters are explained.

Damian Ascanio Hecker
Hendrik Bommer
Garrett Lundegard

Contents

Components and Software

- Windows 10 (other operating systems should be possible, but were not tested)
- MATLAB (2022b) and Simulink
- Simulink Toolboxes:
    - Simulink Desktop Real-Time
    - DSP System Toolbox
    - Signal Processing Toolbox
- BITalino (r)evolution Plugged Kit BLE/BT incl. 2 x sensor cables and EMG electrodes
- BITalino Quickstart Guide
- Open Signals – BITalino data acquisition program for testing the device
- Arduino IDE
- DFRobot Bionic Hand
- DFRduino UNO R3 with IO Expansion Shield
- EMG Control of Robot Hand – Github Repository

System Overview

The EMG Controlled Robot Hand system consists of three main parts:

- BITalino with EMG sensors and electrodes
- Simulink model for data acquisition, processing, and control
- Robot hand driven by an Arduino

The full system diagram and signal flow can be seen in Figure 1. The BITalino device acquires an EMG signal from the actor's forearm muscles. The EMG data is sent to a laptop via a Bluetooth connection. On the laptop, a Simulink model acquires and preprocesses the data and then computes a control signal in real time. This control signal is sent via USB to an Arduino. The Arduino controls the five fingers of a robotic hand by driving a set of five servos, one for each finger, with digital pulse width modulated (PWM) signals. The goal is to allow the actor to open and close the robot hand using only the flexion and extension of their wrist.

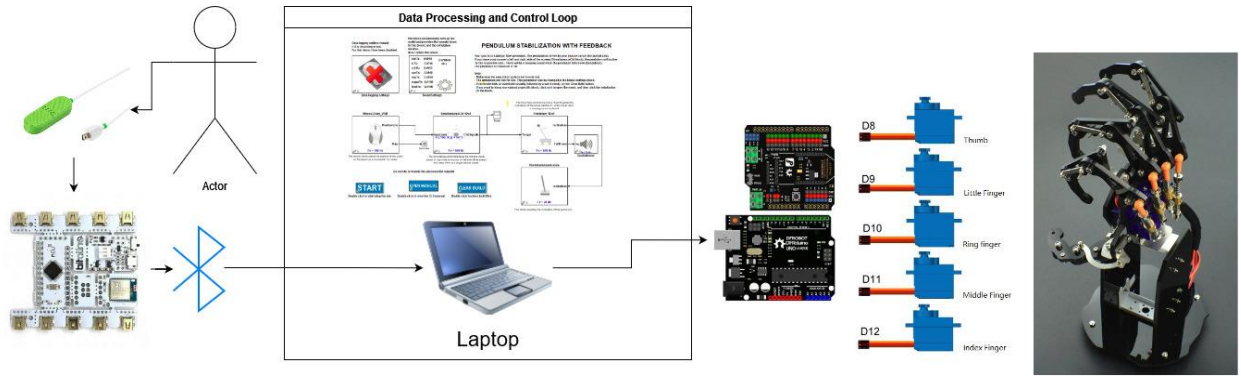


Figure 1 Full System Diagram

Simulink Model – *EMG_System.slx*

The Simulink model is the central hub responsible for acquiring and processing data, computing a control signal, and sending the control signal to the robot hand. The model, shown in Figure 2, consists of a serial USB data acquisition block (“Bitalino Serial To Simulink”), a signal processing block, a control logic block, and a serial USB output to the Arduino (“Serial Control To Arduino”).

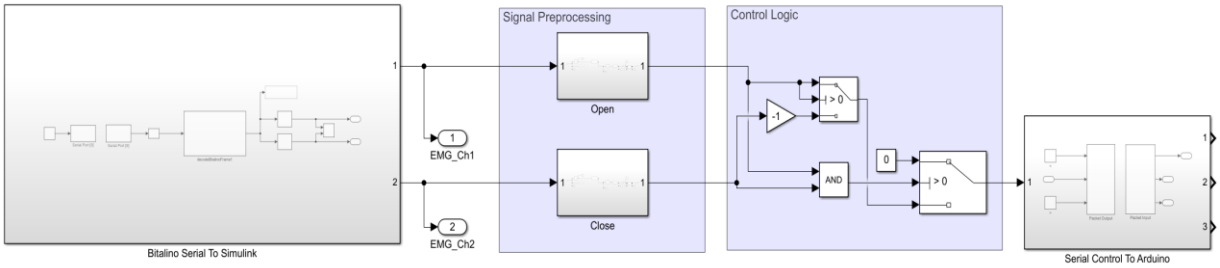


Figure 2 Full System Simulink Model

Signal Acquisition

Signal acquisition takes place via the BITalino over Bluetooth. Two cables with 3 electrodes each (two differential electrodes and one reference electrode) are used. The difference electrodes are placed on the top side (*musculus extensor digitorum*) and the underside (*musculus flexor digitorum superficialis*) of the forearm and the reference electrodes are placed on the bony parts of the wrist (see Figure 3). The ports A0 and A2 on the BITalino are used.

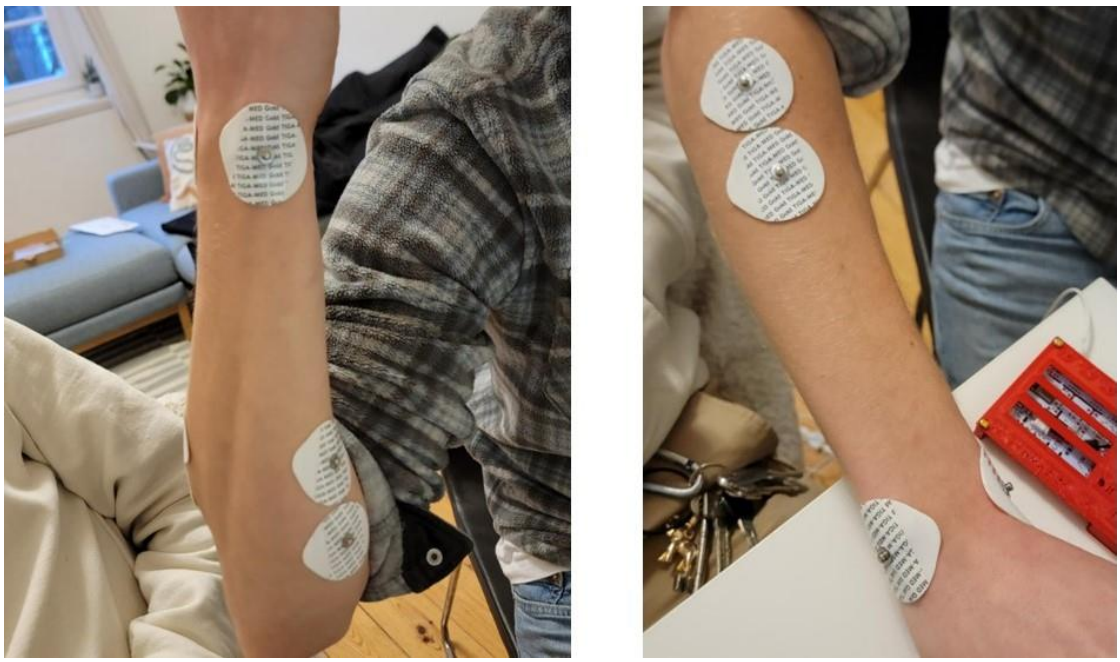


Figure 3 Electrode configuration for the flexor muscles (left) and the extensor muscles (right)

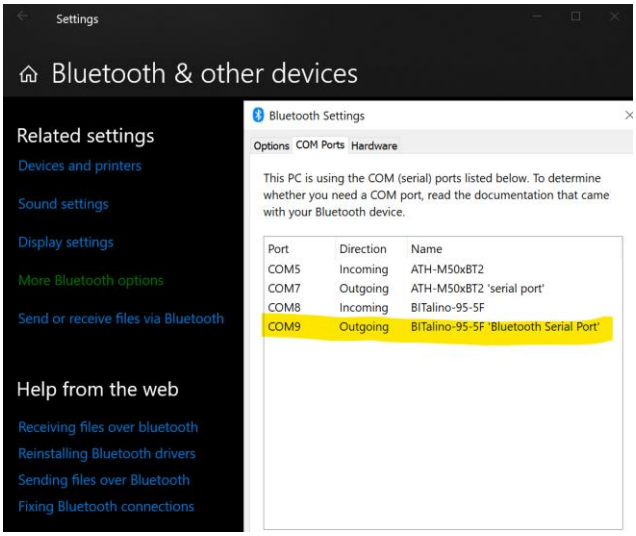
Data acquisition is done in a subsystem in Simulink. Packet Input and Packet Output blocks are used to establish the communication between BITalino and Simulink via a serial port. To initialize the Packet Output block, a new board must first be installed. Here, the COM-port over which the BITalino communicates with the system is to be chosen (see system Bluetooth settings). The COM-port for the BITalino needs to be set to outgoing. The BITalino uses a baud rate of 115200 bps. The initial value of 253 is sent to the BITalino and initializes the “Live” data acquisition mode (indicated by fast flashing of LED). Simulink then receives data in the form of 8-byte data packets which are decoded by a MATLAB script block and the data from channels A0 and A2 are sent as outputs. See the Microcontroller Unit Block Data Sheet for more information on the BITalino data packet format.

Connecting BITalino to Simulink

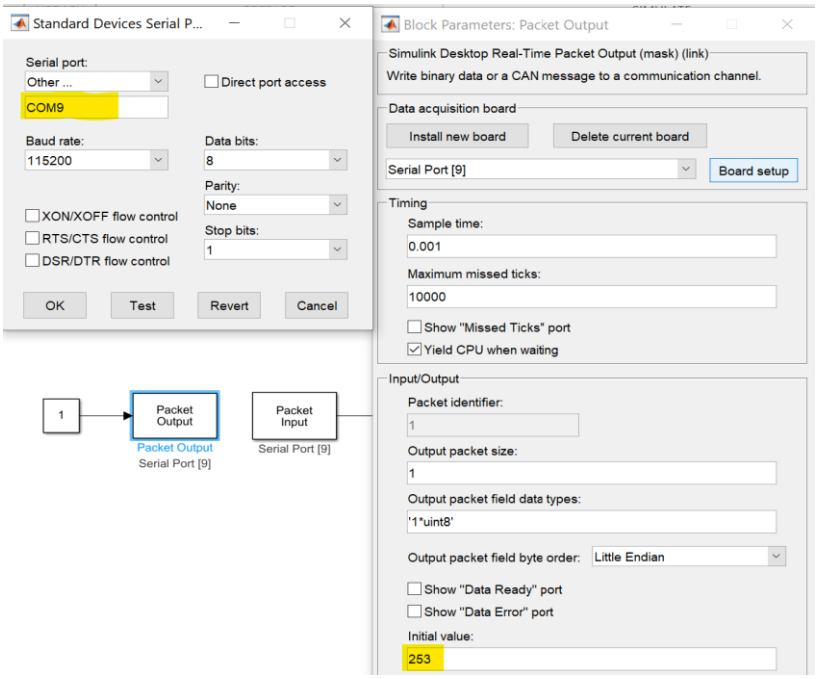
Before proceeding, follow the BITalino Quickstart guide and ensure that the device can connect and stream data to Open Signals. Then, with the BITalino disconnected and powered off:

1. Power on the BITalino and pair it to the laptop via Bluetooth Settings.

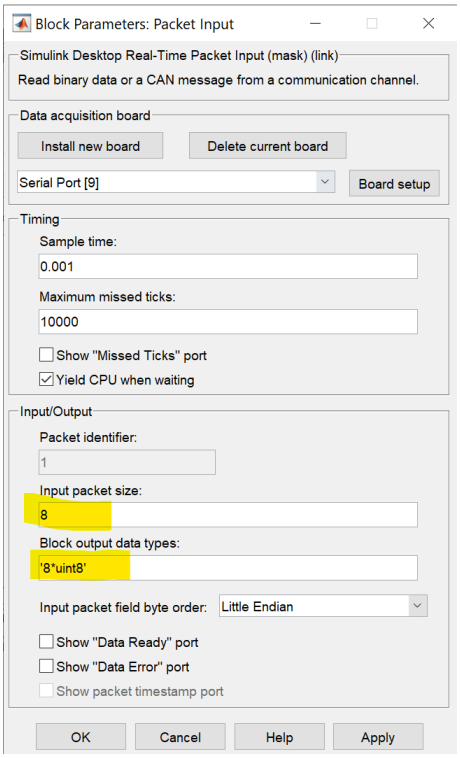
2. Select “More Bluetooth options and find the COM port of the “Outgoing” BITalino device. (Note: while the data will stream from BITalino -> Laptop, the connection and acquisition is initiated by the laptop and therefore “outgoing”… I think)



3. In the Simulink model, ensure the Packet Input/Output blocks match the corresponding port number by double-clicking the block and selecting “Board Setup.” The baud rate used was 115200, however other baud rates should work as well. Also ensure that the “Initial Value” of the Packet Output block is “253,” as this is the code to initiate “Live” data acquisition mode on the BITalino.



4. Ensure that the Packet Input block has the following “Input packet size” and “Block output data types”



Signal Processing

The signal processing is done in two subsystems in Simulink for the open and close movement, respectively. The signal processing chain for a single channel is shown below in Figure 4. Since the muscles are of different size, and therefore EMG signals with different amplitudes are acquired (extension signal was consistently stronger than flexion signal), processing the signal of each movement individually enables the individual adjustment creating similar output signals.

The following contains a walk through one of the subsystems as they are identical in their composition and function. Along the path a few scopes were included to visualize certain processing steps. The transfer function converts the raw signal into a signal in mV around the x-Axis. A Fast Fourier Transformation is also included (but commented out) to determine the cutoff frequencies for the bandpass filter. The bandpass frequency range was chosen to be between 30 - 250 Hz*. Afterwards a moving root-mean-square is computed over the signal using a window size of 100 ms* to smoothen down the signal followed by a data type conversion into values between 0 and 1. The saturation block limits the signal between two values. The values 0.1* and 0.8* were chosen for the lower and upper limit, respectively. Afterwards every value below the lower limit is set to 0 using a switch. The resulting signal has an amplitude between 0.0 - 0.8 but only includes values above 0.1. The signal processed for the closing movement then is flipped around the x-axis and combined with the opening signal. Therefore, the resulting control signal has values between –0.8 and 0.8.

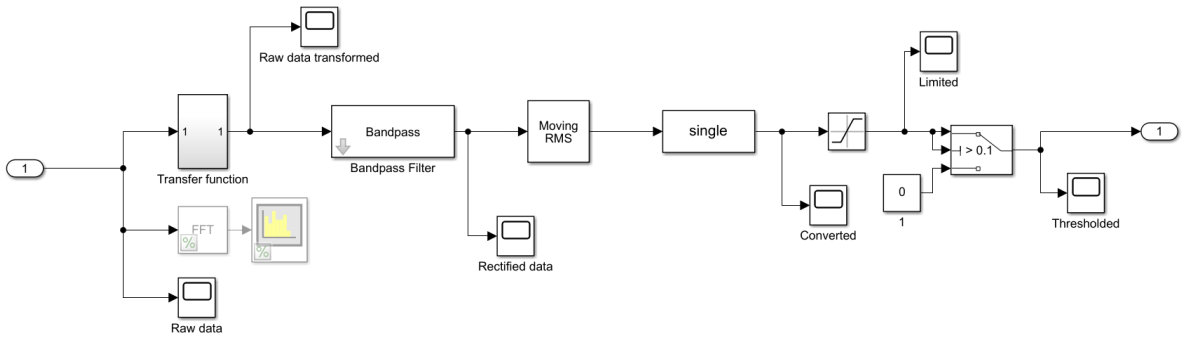


Figure 4 Signal processing chain for a single EMG channel

---

*Adjustable values for further optimization. (Careful: lower limit of saturation block and the switching value need to be identical!)

Control Signal Logic

After preprocessing and thresholding each EMG channel, a simple control system decides whether to send the processed “Open” signal, the processed “Close” signal, or zero. If either signal is greater than 0 while the other is not, that signal is sent directly to the output. If both signals are greater than 0, the “AND” block is triggered and causes 0 to be sent as output. If neither signal is greater than 0, the “Open” signal is sent to the output by default, but it will then be equal to 0 and will have no effect.

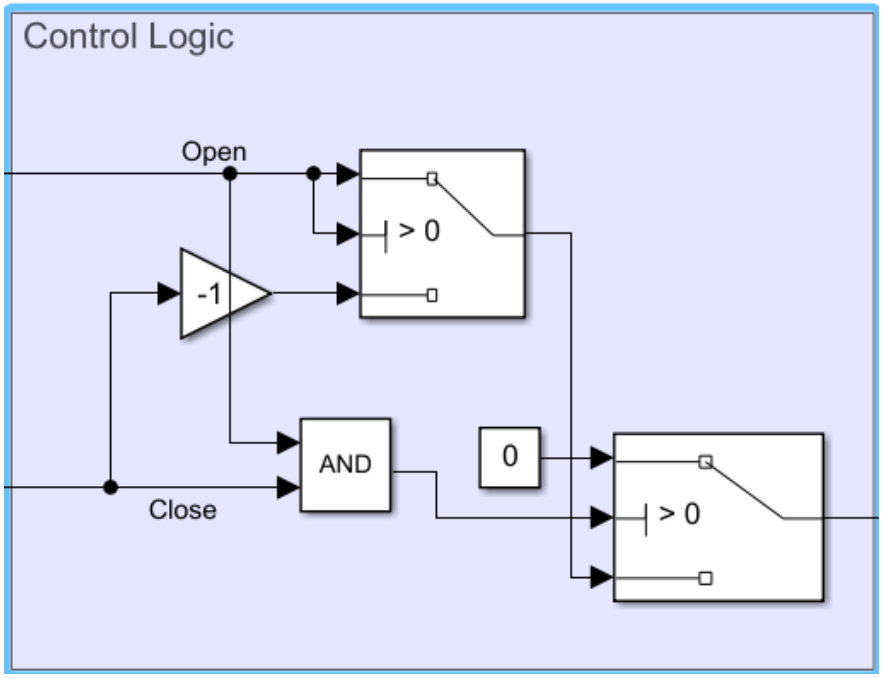


Figure 5 Control Logic

Robot Hand Controller

Hardware

The robot hand is controlled by version of the Arduino Uno produced by DFRobot (DFRduino UNO R3 with IO Expansion Shield), but any Arduino Uno should work. The IO Expansion Shield is useful for providing additional Digital I/O ports with a 3-pin layout (Signal, VCC, GND). The Shield also provides an external port for powering servos. This external power source was not used in this iteration of the project, but it could be useful if the servos don't appear to be getting enough power.

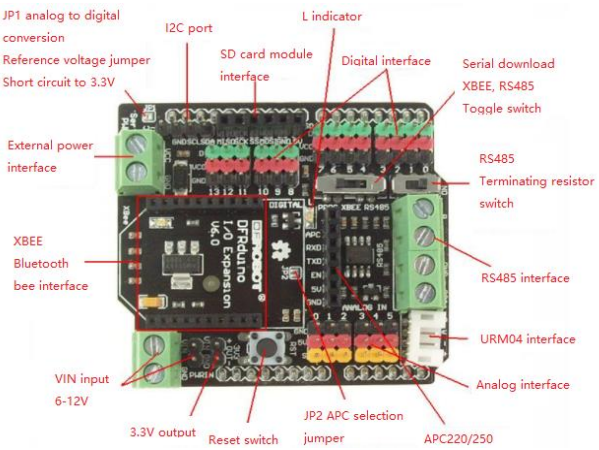


Figure 6 Gravity IO Expansion Shield

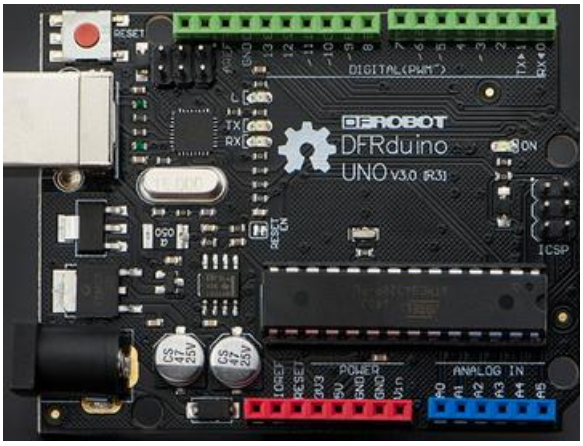


Figure 7 DFRduino UNO

Connect each of the five servos of the robot hand to the IO Shield as shown in Figure 8. The brown wire should be connected to the GND pin, the red wire to the VCC pin, and the yellow wire to the signal ("D") pin.

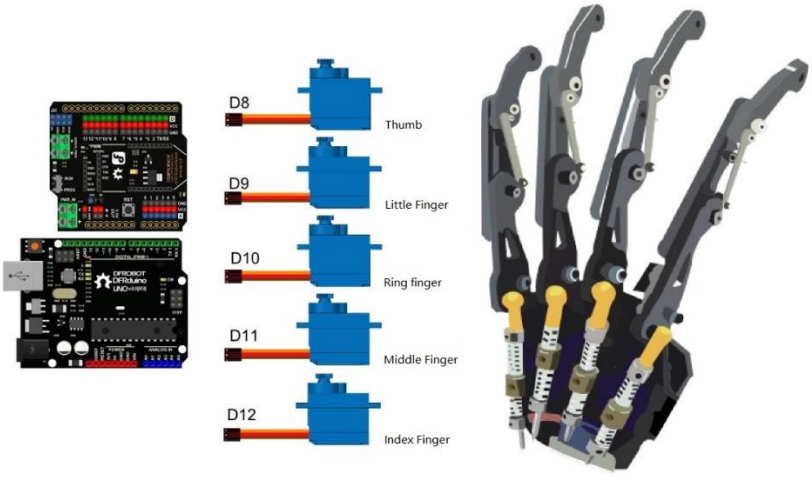


Figure 8 Robot Hand Connection Diagram

Software

The software allowing the Arduino to control the robot hand consist of a main ".ino" file and several C++ program and header files.

RobotHandController.ino

The main function of the program initializes the Serial and Servo Arduino libraries. The Serial and Servo library functionality is implemented as subclasses in C++ files, but because of the way Arduino operates, certain hardware initialization functions are required to be run explicitly in the "setup()" function of the ".ino" file. This includes "Serial.begin()" and "Servo.attach()" (which is called from the subclassed method of "RoboFinger.init()").

After initialization, the "loop()" function runs, continuously checking for new data, and if the received value is positive, the hand will open, and if it is negative, the hand will close.

RoboFinger and RoboHand

RoboFinger is a subclass of the Servo class included with Arduino. This is used to provide an interface to a generic finger.

RoboHand is a class which contains the five fingers, each corresponding to a servo on the robot hand. This class is useful for controlling all five fingers at once with a single function call. Also defined in this class are the maximum and minimum positions of each of the fingers.

SerialSimulink

This class simply hides away the protocol for sending and receiving data from Simulink. The protocol is made to expect messages in this format: "<" + 32-bit Float Value + ">", for a total of 6 bytes.

The Simulink Packet Output block must be formatted accordingly as shown in Figure 9.

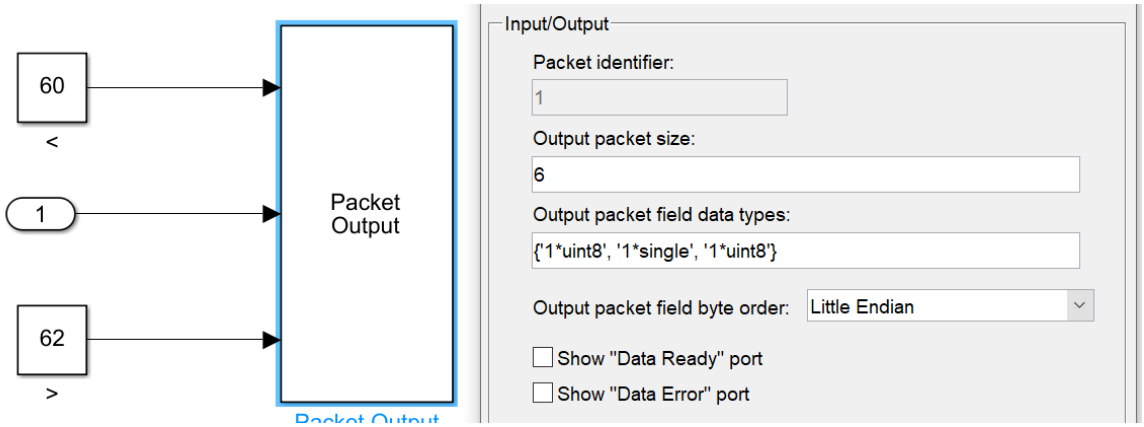
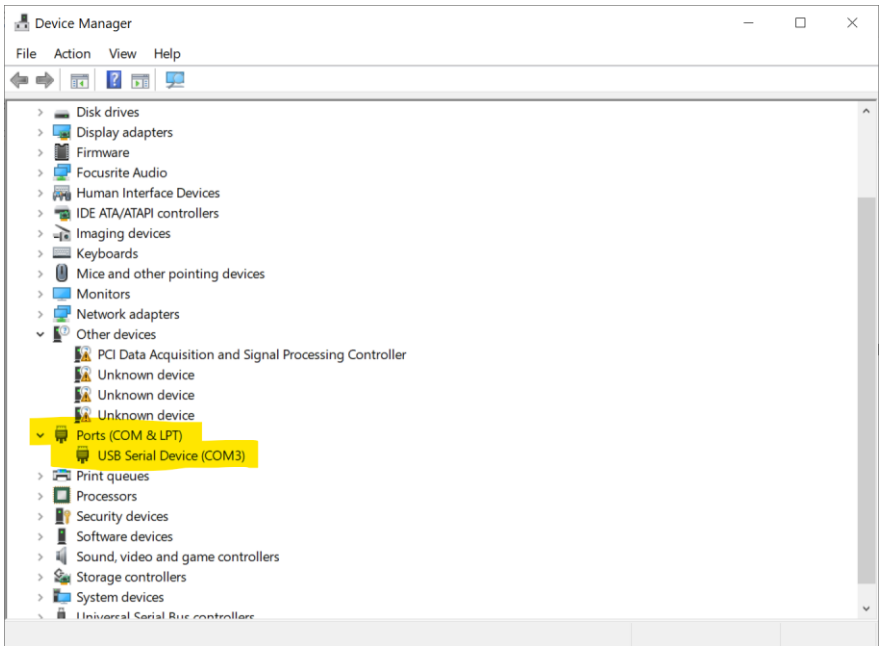


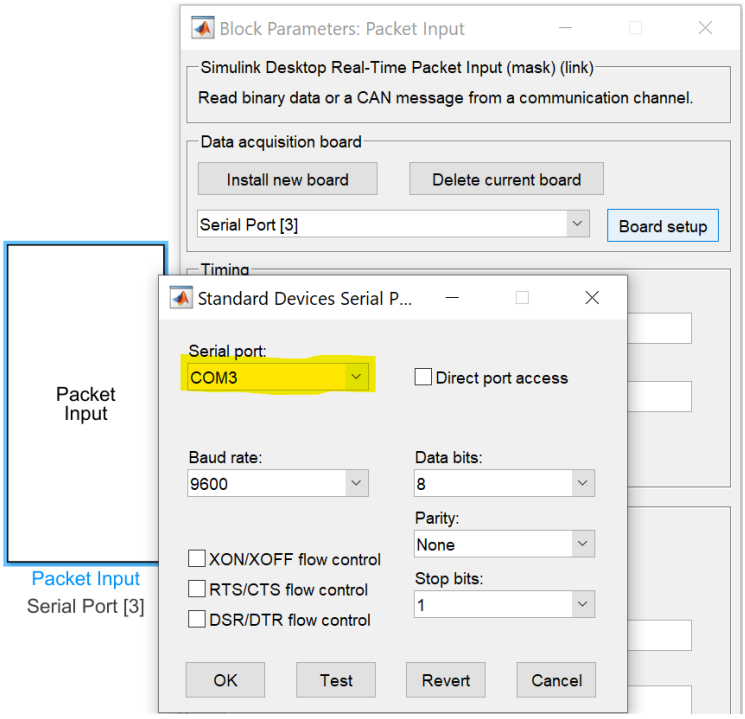
Figure 9 Packet Output data format

Connecting Arduino to Simulink

1. Plug the Arduino into a USB port of the laptop which is running Simulink.
2. Go to “Device Manager -> Ports” on the Windows device and find the port number that Arduino is connected to.



3. In the Simulink model, ensure the Packet Input/Output blocks match the corresponding port number by double-clicking the block and selecting “Board Setup.” The “Baud Rate” should match the value which is set in the main function of the Arduino program (currently 9600).



Further Information

- BITalino User Manual
- Microcontroller Unit Block Data Sheet
- Wiki and Tutorial for Bionic Hand
- Arduino - Servo Library
- Arduino - Serial Library