ESP32/MATLAB Communication Code

A getting started guide

# Overview

## List of core code files

|  |  |  |
| --- | --- | --- |
| ESP32 | MATLAB | Description |
| Message.h | Message.m | Implements a common high level message data class that keeps track of information before and after transmission. |
| SppBluetooth.h | SppBluetooth.m | Serial Bluetooth communication handler class. It takes care of message encoding, sending and receiving. *Note that the ESP code checks for new data, while Matlab uses an event driven structure to reduce processing jitter.* |
| TaskInterface.h |  | Implements an ESP32 task interface. It dictates what functions and variables are required for a task class. |
| Main.cpp |  | Takes care of all practical Bluetooth/scheduling/task call details. |
| board\_type.h |  | Interface for setting and receiving the board type (e.g, Motor Controller) from the eeprom memory. |
| Sheduler.h |  | Implements a basic scheduler. |

## List of example files

|  |  |
| --- | --- |
| ESP32 Demo Files | Description |
| tasks/DemoTask.h | Minimum viable implementation of a task. |
| tasks/DemoMotorControllerTask.h | Implementation of a simulated motor controller. |
| tasks/ DemoSensorBandTask.h | Implementation of a simulated sensor band. |

## High frequency signal sampling

Bluetooth typically have a polling rate of 125 Hz[[1]](#footnote-1) where it processes incoming or outgoing data. This makes it impossible to send and process signals at 1 kHz. To remedy this, these signals are aggregated and transmitted at a lower frequency, e.g., at 50 Hz. I recommended to send signals with a low frequency content at a low frequency to reduce the bandwidth requirement and ultimately reduce variation in network latency. For implementation purposes, the aggregated high frequency samples are transmitted every samples. It looks like this:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Transmission Number | 1 | | | 2 | | | 3 | | | 4 | | |
| High Frequency Samples | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Low Frequency Samples | 1 | 1 | 1 | 4 | 4 | 4 | 7 | 7 | 7 | 10 | 10 | 10 |

Table 1 – Example of sending the high frequency samples every 4th sample. Note that the low frequency samples remain constant throughout a transmission period.

The low frequency transmission rate is calculated by:

## ESP32 Bluetooth name

Each ESP32 have a permanent and unique 12 hexadecimal MAC address that is incorporated into the Bluetooth network name. The mac address is pre-appended to “@Exo-Aider”. Some examples of network names

* 009ABBE350CC@Exo-Aider
* 60FB9912CFA4@Exo-Aider
* 74E80B12CFA4@Exo-Aider

## ESP32 Tasks

An ESP32 task class must inherit the *TaskInterface* class. It contains a few metadata variables that must be set during the *initialization* function.

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Type | Description | Example |
| description | String |  | “Motor Controller” |
| high\_frequency\_sample\_names | List of Strings | Names of the high frequency samples. | {“EMG1”, “EMG2”} |
| low\_frequency\_sample\_names | List of Strings | Names of the low frequency samples. | {“torque”, “IMUx”, “FSR1”} |

The ESP32 task metadata is extracted by MATLAB and used to keep track signals. For sample implementations, see:

* *DemoTask.h*
* *DemoMotorControllerTask.h*
* *DemoSensorBandTask.h*

To make a task available, it must be registered by the “*get\_potential\_tasks”* function in the *“tasks/task\_list.h”* file.

## The Message class

A message contains the following information:

|  |  |  |
| --- | --- | --- |
| Field | Type | Description |
| Command | String | Let’s the receiver know what to do with the message. |
| Numbers | List of floating-point numbers | Number arguments. |
| Strings | List of strings | String arguments. |

## Build in ESP32 commands

See main.cpp for the implementation of the different commands. An overview of the most useful arguments is provided here:

|  |  |
| --- | --- |
| Command | Description/uses |
| ping | Used to check if the MATLAB-ESP32 connection is working. |
| set\_board\_task\_name  get\_board\_task\_name | Sets/gets the board task name. This is the name of the active task. The board must be restarted for any changes to take effect. |
| restart | Restarts the board. |
| get\_sheduler\_periods\_behind | Gets how many samples the scheduler is behind. Useful for debugging purposes. |
| set\_sample\_frequency get\_sample\_frequency | Sets/gets the high frequency sampling frequency. |
| set\_send\_signals\_ratio  get\_send\_signals\_ratio | Sets/gets from the High frequency signal sampling section. |
| set\_send\_signals  get\_send\_signals | Sets/gets if the high and low frequency signals are being transmitted from the ESP32 to Matlab. |
| get\_lf\_signal\_names | Gets a list of the low frequency sample names. |
| get\_hf\_signal\_names | Gets a list of the high frequency sample names. |
| get\_task\_description | Gets the task description. |
| relay | Relays back whatever is send to the ESP32. Useful for determining if the communication protocol works. |

# Common Use Cases

## Getting ESP32 configuration information

On boot the ESP32 transmits its metadata through the serial port. Sample information

|  |
| --- |
| Initializing...  \* Board task name: "demo\_motor\_controller"  \* Bluetooth name: "009ABBE350CC@Exo-Aider"  \* 4 potential tasks: ["demo\_left\_sensor\_band", "demo\_motor\_controller",  "demo\_right\_sensor\_band", "demo\_task"]  \* 23 low frequency signals: ["n", "t", "u", "y", "r", "s3", "s4", "s5", "s6",  "s7", "s8", "s9", "s10", "s11", "s12", "s13", "s14", "s15", "s16", "s17",  "s18", "s19", "s20"]  \* 0 high frequency signals: []  Initialized! |

## Adding a new ESP32 task (Motor Controller, Sensor Band, etc.)

1. Create a new class instance that inherent the *TaskInterface* class. See the demo files in “tasks/*Demo\*.h” for information.*
2. Add an instance of the class to the *get\_potential\_tasks* function in the “tasks/task\_list.h” file.

## Connecting to the ESP32 from MATLAB

To connect to an ESP32 bluetooth connection from Matlab the following command is used:

SppBluetooth(bluetooth\_name, board\_task\_name, buffer\_size)  
Where

bluetooth\_name is the unique name of the ESP32 Bluetooth unit, e.g., 009ABBE350CC@Exo-Aider. This is the Bluetooth name printed during boot as depicted above.

board\_task\_name is the task name that the ESP32 unit should execute, e.g., demo\_motor\_controller. See available tasks during the unit startup.

buffer\_size is the buffer sizes. The SppBluetooth class maintains consistent performance by using ring buffers for storing both messages and signals. With a sampling frequency of and a desired buffer duration of 20 minutes , the buffer size should have elements. This uses around 10 MB per signal. The earliest signals in the buffer are automatically overwritten when the buffer overflows.

Example:

motor\_controller = SppBluetooth(

'009ABBE350CC@Exo-Aider','demo\_motor\_controller', 1200 \* 1000);

motor\_controller.is\_connected % Returns true if connected.

## Controlling the ESP32 from MATLAB

The behavior of the ESP32 can be controlled from MATLAB by sending commands or querying.

B.send(command, numbers, strings)

msg = B.query(command, numbers, strings)  
Where

B is a SppBluetooth object.

command is a command character string, e.g., ‘ping’ or ‘send\_signals’.

numbers is a vector of numbers, e.g., [1, 2, 123] or [] or [true].

strings is a cell array of character strings or a character string, e.g., {‘a’, ‘b’, ‘c’, ‘abc’} or {} or ‘a’.

msg is the answer to the query from the ESP32. It is empty if it timed out.

Example:

B.send('u', 10); % set the input for simulated exoskeleton of the ‘demo\_motor\_controller’ task.

msg = B.query('set\_send\_signals', true); % send signals to MATLAB

assert(msg.numbers(1) == true); % first argument is true of success

## Getting ESP32 signals from MATLAB

Stored signals can be retrieved using the get\_signals function. The available ESP32 signals are printed during boot, e.g., from a previous example: ["n", "t", "u", "y", "r", ... ]. Note that “n” and “t” are non-optional variables, where “n” is the sample number and t is the local ESP32 time for a given sample. To get “u” samples,

signal\_values = B.get\_signals(signal\_idx, sample\_idx)  
Where

signal\_values are the signal values.

signal\_idx specifies which signals to get, e.g., ‘n’ for a single signal and {‘n’, ‘u’, ‘r’} for multiple. The signal names corresponds with the names used for the high and low frequency signals specified in the task during task initialization.

sample\_idx specifies the sample indexes as normal. As end is not available as function arguments, Samples less than 1, , are relative to end, e.g., [-10:0] are is interpreted as [end-10:end]. If B.get\_signals(signal\_idx) are called, then it is the equivalent of using “:”.

Example:

B.sample\_frequency = 1000; % Set sampling frequency.

B.send\_signals\_ratio = 20; % 1000 Hz / 20 = 500 Hz sendings frequency.

B.send\_signals = true; % Enable sending.

pause(1); % Wait a bit for the samples to come in.

signal\_values1 = B.get\_signals('t', -9:0); % Get the last 10 stored samples of t.

signal\_values1 = B.get\_signals({'t', 'u'}, -9:0); % Get the last 10 stored samples of t and u.

signal\_values2 = B.get\_signals({'t', 'u'}, 10:20); % Get the sample 10 to 20 of t and u.

signal\_values3 = B.get\_signal({'t', 'u'}); % Get all stored samples of t and u.

## Logging ESP32 signals from Matlab

The get\_signals function is limited to a few thousands samples per signal, which is inconvenient for longer tests. To overcome this hurdle, the signals can be logged to a file using:

signal\_values = B. start\_log\_signals\_to\_file(filename[, comment])  
Where

filename is the filename of the signals file.

comment is optional and can be used as an identifier, e.g., “subject\_x\_test\_1”.

To stop the logging:

signal\_values = B. stop\_log\_signals\_to\_file()

To read the signals log file:

t = SppBluetooth. load\_signals\_from\_file(filename)  
Where

filename is the filename of the signals file.

t is a table with the recorded signals as columns.

Example:

B.sample\_frequency = 1000;

B.send\_signals\_ratio = 40;

B.send\_signals = true;

B.start\_log\_signals\_to\_file('my\_log\_file.dat');

% Execute the experiment

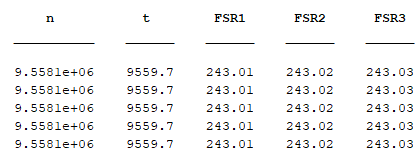
pause(10);

% Stop the experiment

B.stop\_log\_signals\_to\_file();

t = SppBluetooth.load\_signals\_from\_file('my\_log\_file.dat');

Example of t:



## List of important MATLAB commands for ESP32-MATLAB communication

Let B denote a connected SppBluetooth object.

|  |  |
| --- | --- |
| Property/function | Description/Behavior |
| b = B.send\_signals B.send\_signals = b | Gets/sets the ESP32 send\_signal variable. |
| msg = B.ping() | Pings the ESP32. Msg.numbers contains local time of the ESP32. |
| B.send\_signals\_ratio = n  n = B.send\_signals\_ratio | Gets/sets the ESP32 send\_signals\_ratio variable. |
| b = B.is\_connected | Checks if the connection is still alive. |
| B.get\_signals | Get stored signals. |
| B.start\_log\_signals\_to\_file | While send\_singal is true, all received signals are written to a file. |
| B.stop\_log\_signals\_to\_file | Stop writing received signals to a file. |
| t = B.load\_signals\_from\_file t = SppBluetooth.load\_signals\_from\_file | Read a saved signals file as a table. |

## Emulated exoskeleton control system

A small test is performed to get a feeling for the real-time performance of a simulated closed loop exoskeleton control system depicted on Figure 1. To create a realistic data processing load on Matlab, two sensor armbands are simultaneous transmitting their data to Matlab for processing. Figure 2 depicts the units used for the test setup. The devices used for the test includes:

|  |  |
| --- | --- |
| Unit | Description/function |
| Matlab | Controller and Bluetooth master. |
| Motor Controller (ESP32 #1) | Board name: “demo\_motor\_controller”. Simulates a first order transfer function. |
| Left Sensor Band (ESP32 #2) | Board name: “demo\_left\_sensor\_band”. |
| Right Sensor Band (ESP32 #3) | Board name: “demo\_right\_sensor\_band”. Same function as the Left Sensor Band above. |

See the SppBluetooth\_test.m for an example.

|  |  |
| --- | --- |
| Figure 1 - Closed loop simulation of exoskeleton | Figure 2 - Test setup configuration. Black arrows indicate normal communication, while red arrows indicate periodic sending of signals when its local "send\_signals" variable is true. |

1. I’ve got this number from Bluetooth mouse specifications, but it corresponds nicely with my own findings. [↑](#footnote-ref-1)