MAVLink Documentation for PX4 and Simulink Implementations

Andrei Alberto Cuenca

Version 2, February 2020

Introduction to Mavlink

MAVLink is a Serial communication protocol used for small unmanned vehicles. It is mostly used for communication between Ground Control Stations and Unmanned vehicles, and it is possible to identify what vehicle and what subsystem of the vehicle is sending information.

The libraries and blocks developed in this work allows communication between any PC running Simulink in Real time, PX4 running simulink generated code with the PX4 code embedded support Package and PC running Ground control software as Mission planner.

Message Format

The protocol defines a large set of messages and those are identified in the files contained in the MAVlink V1.0 library (https://github.com/mavlink/c_library_v1).The Message format is presented in the Table 1. and it will be the key information to create a package:

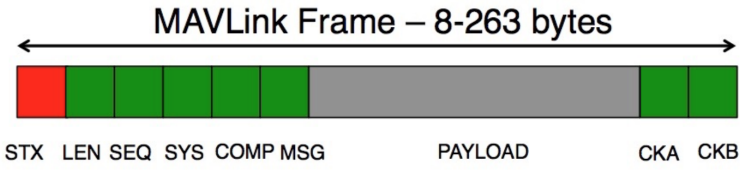

Figure 1: Mavlink Frame

Table 1: Mavlink package description

Field name	Byte Index	Value	Definition
Packet Start	0	0xFE	Denotes the start of a new packet
Payload length	1	0-255	Length in bytes of the following payload
Packet Sequence	2	0-255	Count up of each time package has been sent
System ID	3	0-255	ID of the Sending system
Component ID	4	0-255	ID of the Sending component
Message ID	5	0-255	ID of the msg. The MSG ID defines what the payload means and how it should be decoded
Payload (Data)	6+n	0-255 Bytes	Data of the Message
Checksum	6+n+1 to 6+n+2		ITU x.25/SAE AS-4 hash CRC-16 excluding start packet 0xFE.

The file common.xml (https://github.com/mavlink/mavlink/blob/master/message_definitions/v1.0/common.xml)contains the definition of each of the messages for its identification as long with the characteristics of the data in the payload.

Mavlink Encode and Decode Blocks for Simulink integration

This section will describe the structure of the designed blocks for encode and decode and a method to integrate the developed MAVlink protocol in the Simulink environment. These blocks can be used in any application that requires code generation.

Simulink Blocks to Encode MAVLink messages

NOTE: This method is supported for embedded code generation in Simulink. It is compatible with PX4 code building.

The Encoding procedure uses basics blocks of Simulink to create the Stream of binaries following the Mavlink format. This method requires a medium knowledge in binary or HEX operations to understand how a MAVLink package is composed and how is its serialization. Some blocks for specific messages are already created but further work is required to create new encoding blocks for new messages. This document will explain the structure of the blocks and how to create a new encoding block for a new or custom required message.
The structure of the header is described as follows:

- **Packet Start:** A Mavlink message starts with the number 254 or 0xFE in hex.
- **Payload Length:** Specifies the number of bytes that the payload witll have. This number can be calculated by adding the number of bytes of each element of the payload. The number of bytes of each element depends on its format. This is referenced in the Table.2
- **Packet Sequence:** The Encoding procedure is set up to count how many times the packet has been sent until 254 and resets. This gives a clue to the receiver if all sequential packets where received or a packet was lost or discarded in the process.
- **System ID:** This number is the Identification Number of the system that is encoding the packet. This helps to identify the source of the packet in a network of multiple devices communicating using the Mavlink protocol.
- **Component ID:** This number is the identification number of a component within a system. It helps to identify what component of a system emitted the packet if various component on the same devices use Mavlink.
- **Message ID:** This is the Identification Number of the type of message that is going to be created. Mavlink has different messages each one with its own ID like local position,attitude,RC raw, hearth beat,IMU raw, etc

The structure of the payload varies depending of the message that is going to be created. The main considerations here are the number of elements and the format of each elements. This will define the number of bytes.

After the payload, two additional bytes are attached by a process called Cyclic Redundancy check (CRC). The CRC is and error detecting algorithm used to guarantee the integrity of the packet by the receiver. This process will be explained in detail in the next section.

Create Encode message block for specific message in Simulink

1. Create the mavlink header. It follows the same structure for every message but the **payload length** and **Message ID** are unique values for each message. These values can be found in the respective .h header file of the message in the mavlink library. For example, this is how the mavlink_msg_attitude.h looks like:

```
#define MAVLINK_MSG_ID_ATTITUDE 30

MAVPACKED(
typedef struct __mavlink_attitude_t {
 uint32_t time_boot_ms; /*< Timestamp (milliseconds since system boot)*/
 float roll; /*< Roll angle (rad, -pi..+pi)*/
 float pitch; /*< Pitch angle (rad, -pi..+pi)*/
 float yaw; /*< Yaw angle (rad, -pi..+pi)*/
 float rollspeed; /*< Roll angular speed (rad/s)*/
 float pitchspeed; /*< Pitch angular speed (rad/s)*/
 float yawspeed; /*< Yaw angular speed (rad/s)*/
}) mavlink_attitude_t;

#define MAVLINK_MSG_ID_ATTITUDE_LEN 28
#define MAVLINK_MSG_ID_ATTITUDE_MIN_LEN 28
#define MAVLINK_MSG_ID_30_LEN 28
#define MAVLINK_MSG_ID_30_MIN_LEN 28

#define MAVLINK_MSG_ID_ATTITUDE_CRC 39
#define MAVLINK_MSG_ID_30_CRC 39
```

   From here it is posible to extract all the information needed:

   The MAVLINK_MSG_ID_ATTITUDE contains the **Message ID = 30**.

   The MAVLINK_MSG_ID_ATTITUDE_LEN contains the **Payload Lenght = 28**.

   The MAVLINK_MSG_ID_ATTITUDE_CRC contains the **CRC_EXTRA Attitude = 39**.

   Replace these values in the header section of the simulink block.

2. Create the mavlink payload of the respective message. The related .h header file of the message also contains information of the type of each elements an the order they should have in the payload package so the decoding process follows the same format.

   Another source where it is posible to identify the Name of the message with its ID, Payload length, Number of elements and formats is the Mavlink Development website (https://mavlink.io/en/messages/common.html)

   For simulink, refer to the next table to handle the format of the elements:

Table 2: Datatype convertion

Mavlink	Simulink	# Bytes
uint8_t	uint8	1
int8_t	int8	1
uint16_t	uint16	2
int16_t	int16	2
uint32_t	uint32	4
int32_t	int32	4
uint64_t	double	8
float	single	4
char	uint8	4

   Once the Header and payload are defined, the CRC bytes have to be calculated. In the Mavlink protocol, an extra byte is added after the payload before the CRC calculation. This byte is called **Extra CRC** and is meant to avoid confutions between different messages.

3. The Header and the payload Are muxed together making sure all of the elements enter as uint8 format, it means, in bytes. For this task, the block **bytpe Pack** is the most convenient since it converts every vector on its input to uint8 elements. Refer to the figure 3

4. The **CRC Extra** is added after, at the end of the Mux (The mux vector of last point) This is constant value uint8 with the value of the **CRC_Extra** related to the message, which in this case is 39. This value is used by the CRC to generate the correct two checksum bytes. Refer to the figure 3.

5. The CRC section requires two specific changes since each message is different in size. The input to this block is a vector of bytes with size [1:n] (n elements). The first selector takes out the first element of the vector, it means, the Packet Start byte (0xFE), becoming [2:n]. The last selector switches the order of the last two bytes and deletes the third last item of the vector, so the new order is [2:n-3 n n-1].

   The last two bytes ([n n-1]) are the result of the CRC block and the byte deleted ([n-2]) correspond to the **CRC extra** that is not longer required.

   Example: For the attitude message the header has 6 elements + 28 elements from the payload + 1 of the CRC_EXTRA number. The input to the CRC block would have 35 elements. The first selector would be size [2:35] (it deletes the start byte). The CRC generator adds two extra bytes to the stream at the end but inverted as they should be. The last selector arrange the position of these last two bytes and deletes the CRC_EXTRA that was added before the CRC calculation, so now the vector is [2:34 37 36].

6. The last step is to add again the start packet at the beginning of the stream before it is sent over a serial port. Now the packet is ready to be sent. An example is attached to the guide.

An explanation in detail on how the CRC procedure is implemented in the mavlink protocol is presented in the document for debugging purposes. This is useful if the protocol has to be developed in other languages.

Cyclic redundancy check in Mavlink protocol (CRC)

The communication process is the transmission of zeros and ones and in the process, errors can be generated. The CRC is a binary algorithm that checks the integrity of the full stream and is performed by the transmiter and also the receiver.

The transmitter calculates the CRC bytes of its packet and sends the result contained at the end of the packed to the receiver. Here, the receiver extracts the packet from the stream without the CRC bytes from the transmiter, and calculates the CRC bytes with the information received. Then the CRC bytes received are compared with the CRC bytes calculated to check integrity. If there is a match, the packet does not contain any error, if not, it is discarded

The mavlink protocol uses standard CRC-16-CCITT with initial value 0xFFFF. The polynomial that is used is $x^{16} + x^{12} + x^5 + 1$ with the binary representation [0001 0000 0010 0001] and Heximal representation 0x1021.

First, the bytes of the data (without including the start byte) are inverted and a First CRC is calculated using the initial value 0xFFFF with the polinomial 0x1021. Then the **CRC Extra** is inverted and a CRC is calculated but the initial value is the result of the last computed CRC. The result is a new will give 2 bytes (16 bits) that needs to be inverted back. This two bytes correspond to the Final CRC.

As an example to understand the CRC verificacion, consider next heartbeat message packet:

Table 3: Heartbeat Mavlink Message

Byte #	**Byte Value (HEX)**	**Byte Value (DEC)**	**Content**
0	0xFE	254	Packet Start
1	0x09	9	Payload Length
2	0x40	64	Packet Sequence
3	0x01	1	System ID
4	0x01	1	Component ID
5	0x00	0	Message ID
6	0x06	6	Payload Counter Byte 1
7	0x00	0	Payload Counter Byte 2
8	0x00	0	Payload Counter Byte 3
9	0x00	0	Payload Counter Byte 4
10	0x0D	13	Payload Mav. type
11	0x03	3	Payload Mav. Autopilot
12	0x59	89	Payload Base Mode
13	0x03	3	Payload System Status
14	0x03	3	Payload Mav. version
15	0x94	148	CRC 1
16	0x22	34	CRC 2

As defined in the payload length, there are 9 bytes labeled as a payload for this message. Notice that the **CRC Extra** Never appears on the mavlink structure. This value is only used for purposes of calculating the CRC bytes, but is never included in the final package.

The first CRC is calculated taking the Bytes 1 to 14 and the initial condition as 0xFFFF. The result are 2 bytes that are now the initial condition for a second CRC that takes the CRC extra as the input. This results on two CRC bytes that are attached to the message.

EXAMPLE: Encode Heartbeat and Attitude Msg and transmit data from a PX4

The following example defines a Heartbeat message and reads the angle rates and the Euler angles from the PX4 sensors in order to construct the attitude message. These packages will be sent through the Telemetry 1 port of a Pixhawk. This example makes use of the **PX4 embedded coder support package** of Matlab to create a PX4 compilable model

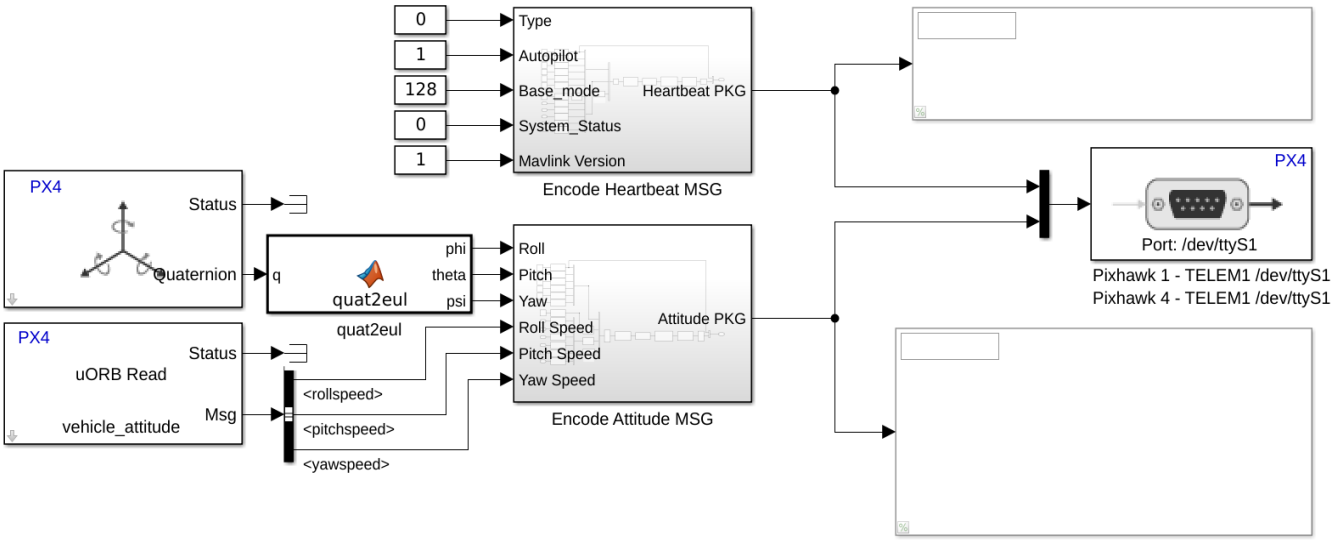

Figure 2: Simulink example: Create, encode and send Mavlink packages in a PX4

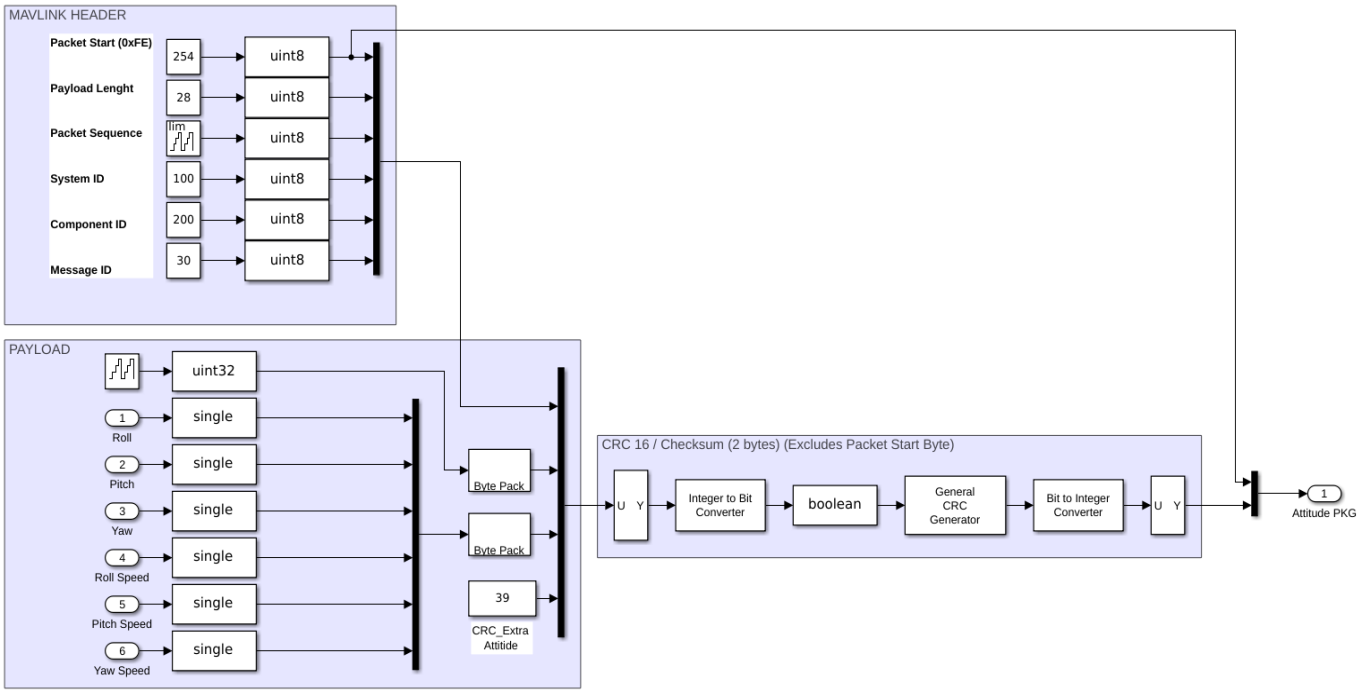

Figure 3: Simulink Mavlink encoding structure for a specific message

Both encode blocks follows the same structure presented at the beginning of the section. Each block uses the same header with the respective ID number and and creates its own payload following the structure presented in the respective header (.h files).

The blocks handle the CRC as it should so, it is not necessary to modify any parameter of the CRC block.

Create Decode message block for specific message

The mavlink decoder block is a custom matlab function that follows the next architecture:

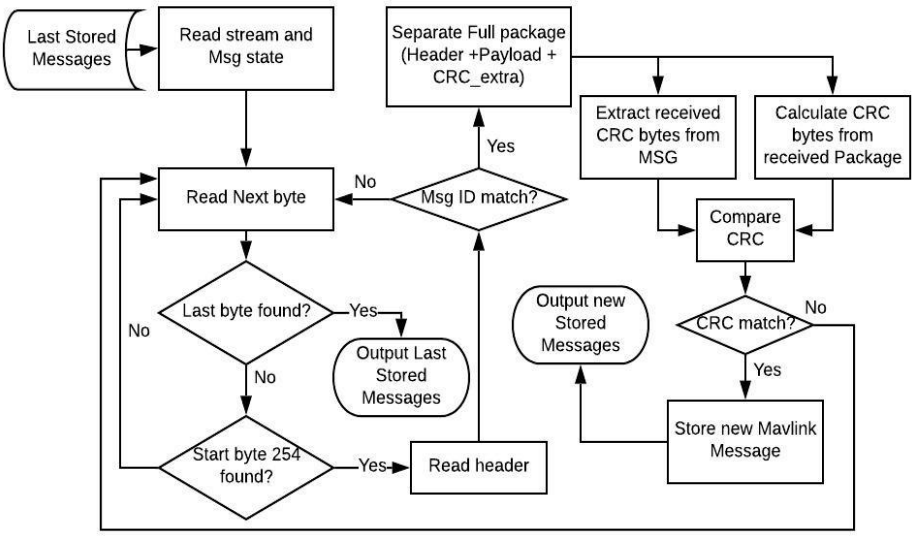

Figure 4: Mavlink Decode algorithm implemented

The decode block is a matlab function, and it is necessary to define the inputs and outputs. The outputs correspond to the messages required to be decoded and the inputs are the Stream of data and the last known stored values of the messages decoded.

The decode structure for a specific message is written as follows in the matlab code:

```
1
2  %(ADD HERE THE MESSAGE TO BE DECODED IN THE INPUT AND OUTPUT)
3  function [heartbeat,Attitude] = MavlinkDecoder(heartbeat_in,Attitude_in,Stream)
4  %% The following block Decodes the specified Mav packages listed in Outputs.
5  % The input for this function is the stream received from the serial
6  % comunication and the values from the last sample time.
7
8  %% Initial definitions ----------------------------------------------------
9  %(ADD HERE A FLAG VARIABLE FOR THE NEW MESSAGE)
10 SM=0;        %Flag Start byte Detected
11 a=1;         %Count for bytes
12 FHB=0;       %Flag HeartBeat Msg Saved
13 FATT=0;      %Flag Attitude Msg Saved
14
15 Size=length(Stream);
16
17 %Call last value stored for each Message
18 % (ADD HERE MESSAGE INPUT FOR INITIAL CONDITIONS)
19 heartbeat=heartbeat_in;
20 Attitude=Attitude_in;
21
22 %Define CRC conditions and generator--------------
23 %(ADD HERE THE CRC CONSTANT VALUE FOR THE SPECIFIC MESSAGE TO BE DECODED)
24 CRC_HB=uint8(50);   %CRC_Extra value for Heartbeat
25 CRC_ATT=uint8(39);  %CRC_Extra value for Attitude
26 CRC_def=comm.CRCGenerator('z^16 + z^12 + z^5 + 1','DirectMethod',true,...
27     'InitialConditions',[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1],...
28     'ReflectInputBytes',true,'ReflectChecksums',true);
```

For each Message to be decoded a conditional if has to be added to the main comparator where takes place the identification of the message ID once a start packet (254) is detected. Since the ID, payload and size of the messages is already known, the only thing needed is to compare the header to verify if the message detected is the one expected and then check its integrity comparing the CRC bytes received with ones calculated.

```
%% Decode process ----------------------------------------------------------
while a<230
    if (SM==0 && Stream(a)==254)

        %Tells if a message is being read
        SM=1;
        %Read header ID 0xFE (254) if detected
        PL=Stream(a+1);     %Read Payload
        SysID=Stream(a+3);  %Read System ID
        CompID=Stream(a+4); %Read ComponentID
        ID=Stream(a+5);     %Read Message ID

        % Decode heartbeat-------------------------------------------------
        % Check if message started and compare ID, payload matches and flag is empty
        if (SM== 1 && ID==0 && PL==9 && FHB==0) %MAIN COMPARATOR--------------------------
            %Reads the Package and adds the CRC_extra
            PKG=[Stream(a+1:a+14)' CRC_HB];

            %Extract CRC bytes from incoming MSG
            CRC1_R=Stream(a+15);
            CRC2_R=Stream(a+16);

            %Convert Package in binary stream
            PKG=de2bi(PKG,8,'left-msb');

            %Prealocatting binary Stream
            PKG_bin=zeros(1,15*8);
            PKG_bin(1:8)=PKG(1,:);
            for b=2:15
                PKG_bin(b*8-7:b*8)= PKG(b,:);
            end

            %Calculates CRC with received Package
            CRC=CRC_def(logical(PKG_bin'));

            CRC=CRC';
            CRC2_C=bi2de(CRC(end-15:end-8),'left-msb');
            CRC1_C=bi2de(CRC(end-7:end),'left-msb');

            %Compares CRC received with Calculated for MSG integrity
            if (CRC1_R==CRC1_C && CRC2_R==CRC2_C)
                heartbeat=Stream(a+6:a+14);
                SM=0;
                FHB=1;
            else
                SM=0;
                a=a+1;
            end
        else
            a=a+1;
            SM=0;
        end %END MAIN COMPARATOR-----------------------------------------------------------
    else
            a=a+1;
            SM=0;
    end
end
```

For each case, if a start packet, ID and payload were successfully detected and the CRC received and calculated matched, the message is saved for its used. If any step fails the algorithm keeps checking the streaming for a new packet start byte.

EXAMPLE: Receive and Decode a Heartbeat and Attitude Msg in a PX4

The following example receives the data being sent from the previous example from a PX4 in another PX4. The receiver PX4 can be connected in external mode to verify the data.

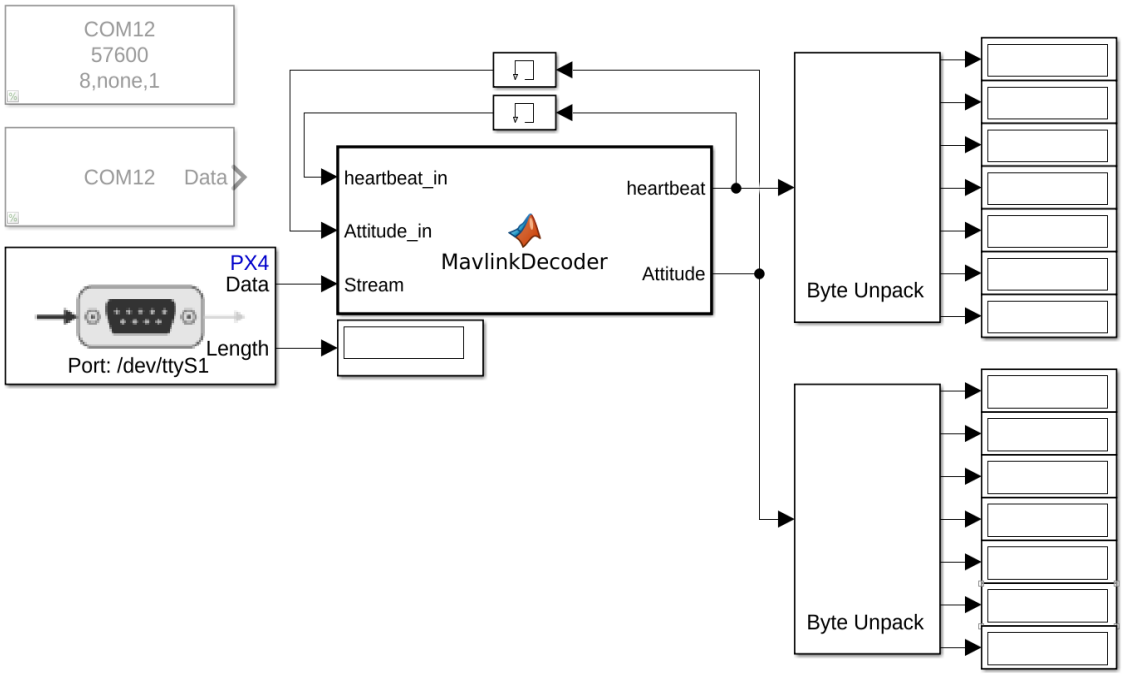

Figure 5: Simulink example:Receive and Decode a Heartbeat and Attitude Msg in a PX4