#include <SoftwareSerial.h>

#include "mavlink.h"

#define RX\_PIN 0 //used for Mavlink, reads input from Pixhawk

#define TX\_PIN 1 //used for Mavlink, transmits to Pixhawk

SoftwareSerial MavSerial = SoftwareSerial(RX\_PIN, TX\_PIN);

unsigned long previousMillisMAVLink = 0;     // will store last time MAVLink was transmitted and listened

unsigned long next\_interval\_MAVLink = 1000;  // next interval to count

const int num\_hbs = 60;                      // # of heartbeats to wait before activating STREAMS from Pixhawk. 60 = one minute.

int num\_hbs\_pasados = num\_hbs;

void setup()

{

  Serial.begin(57600);

  while (!Serial)

  {

    ; // wait for port to connect.

  }

  MavSerial.begin(57600);

  while (!MavSerial)

  {

    ; // wait for port to connect.

  }

  Serial.println("Ready");

if (MavSerial.available() > 0 )

  {

    Serial.print("Num Bytes: ");

    Serial.println(MavSerial.available());

  }

}

void loop()

{

  int sysid = 1;                   ///< ID 20 for this airplane. 1 PX, 255 ground station

  int compid = 158;                ///< The component sending the message

  int type = MAV\_TYPE\_GROUND\_ROVER;   ///< This system is an airplane / fixed wing

  // Define the system type, in this case an airplane -> on-board controller

  // uint8\_t system\_type = MAV\_TYPE\_FIXED\_WING;

  uint8\_t system\_type = MAV\_TYPE\_GENERIC;

  uint8\_t autopilot\_type = MAV\_AUTOPILOT\_INVALID;

  uint8\_t system\_mode = MAV\_MODE\_PREFLIGHT; ///< Booting up

  uint32\_t custom\_mode = 0;                 ///< Custom mode, can be defined by user/adopter

  uint8\_t system\_state = MAV\_STATE\_STANDBY; ///< System ready for flight

  // Initialize the required buffers

  mavlink\_message\_t msg;

  uint8\_t buf[MAVLINK\_MAX\_PACKET\_LEN];

  // Pack the message

  //mavlink\_msg\_heartbeat\_pack(sysid,compid, &msg, type, autopilot\_type, system\_mode, custom\_mode, system\_state);

  mavlink\_msg\_heartbeat\_pack(1,0, &msg, type, autopilot\_type, system\_mode, custom\_mode, system\_state);

  // Copy the message to the send buffer

  uint16\_t len = mavlink\_msg\_to\_send\_buffer(buf, &msg);

  // Send the message with the standard UART send function

  // uart0\_send might be named differently depending on

  // the individual microcontroller / library in use.

  unsigned long currentMillisMAVLink = millis();

  if (currentMillisMAVLink - previousMillisMAVLink >= next\_interval\_MAVLink) {

    // Guardamos la última vez que se cambió el modo

    previousMillisMAVLink = currentMillisMAVLink;

#ifdef SOFT\_SERIAL\_DEBUGGING

    pxSerial.write(buf,len);

    //mySerial.println("Ardu HB");

#else

    Serial.write(buf, len);

#endif

    //Mav\_Request\_Data();

    num\_hbs\_pasados++;

    if(num\_hbs\_pasados>=num\_hbs) {

      // Request streams from Pixhawk

#ifdef SOFT\_SERIAL\_DEBUGGING

      mySerial.println("Streams requested!");

#else

      Serial.println("Streams requested!");

#endif

      Mav\_Request\_Data();

      num\_hbs\_pasados=0;

    }

  }

comm\_receive();

//Serial.print("Jai Balayya");

}

void Mav\_Request\_Data()

{

  mavlink\_message\_t msg;

  uint8\_t buf[MAVLINK\_MAX\_PACKET\_LEN];

  // STREAMS that can be requested

  /\*

   \* Definitions are in common.h: enum MAV\_DATA\_STREAM

   \*

   \* MAV\_DATA\_STREAM\_ALL=0, // Enable all data streams

   \* MAV\_DATA\_STREAM\_RAW\_SENSORS=1, /\* Enable IMU\_RAW, GPS\_RAW, GPS\_STATUS packets.

   \* MAV\_DATA\_STREAM\_EXTENDED\_STATUS=2, /\* Enable GPS\_STATUS, CONTROL\_STATUS, AUX\_STATUS

   \* MAV\_DATA\_STREAM\_RC\_CHANNELS=3, /\* Enable RC\_CHANNELS\_SCALED, RC\_CHANNELS\_RAW, SERVO\_OUTPUT\_RAW

   \* MAV\_DATA\_STREAM\_RAW\_CONTROLLER=4, /\* Enable ATTITUDE\_CONTROLLER\_OUTPUT, POSITION\_CONTROLLER\_OUTPUT, NAV\_CONTROLLER\_OUTPUT.

   \* MAV\_DATA\_STREAM\_POSITION=6, /\* Enable LOCAL\_POSITION, GLOBAL\_POSITION/GLOBAL\_POSITION\_INT messages.

   \* MAV\_DATA\_STREAM\_EXTRA1=10, /\* Dependent on the autopilot

   \* MAV\_DATA\_STREAM\_EXTRA2=11, /\* Dependent on the autopilot

   \* MAV\_DATA\_STREAM\_EXTRA3=12, /\* Dependent on the autopilot

   \* MAV\_DATA\_STREAM\_ENUM\_END=13,

   \*

   \* Data in PixHawk available in:

   \*  - Battery, amperage and voltage (SYS\_STATUS) in MAV\_DATA\_STREAM\_EXTENDED\_STATUS

   \*  - Gyro info (IMU\_SCALED) in MAV\_DATA\_STREAM\_EXTRA1

   \*/

  // To be setup according to the needed information to be requested from the Pixhawk

  const int  maxStreams = 2;

  const uint8\_t MAVStreams[maxStreams] = {MAV\_DATA\_STREAM\_EXTENDED\_STATUS, MAV\_DATA\_STREAM\_EXTRA1};

  const uint16\_t MAVRates[maxStreams] = {0x02,0x05};

  for (int i=0; i < maxStreams; i++) {

    /\*

     \* mavlink\_msg\_request\_data\_stream\_pack(system\_id, component\_id,

     \*    &msg,

     \*    target\_system, target\_component,

     \*    MAV\_DATA\_STREAM\_POSITION, 10000000, 1);

     \*

     \* mavlink\_msg\_request\_data\_stream\_pack(uint8\_t system\_id, uint8\_t component\_id,

     \*    mavlink\_message\_t\* msg,

     \*    uint8\_t target\_system, uint8\_t target\_component, uint8\_t req\_stream\_id,

     \*    uint16\_t req\_message\_rate, uint8\_t start\_stop)

     \*

     \*/

    mavlink\_msg\_request\_data\_stream\_pack(2, 200, &msg, 1, 0, MAVStreams[i], MAVRates[i], 1);

    uint16\_t len = mavlink\_msg\_to\_send\_buffer(buf, &msg);

#ifdef SOFT\_SERIAL\_DEBUGGING

    pxSerial.write(buf,len);

#else

    Serial.write(buf, len);

#endif

  }

}

void comm\_receive() {

  mavlink\_message\_t msg;

  mavlink\_status\_t status;

  // Echo for manual debugging

  //Serial.println("---Start---");

#ifdef SOFT\_SERIAL\_DEBUGGING

  while(pxSerial.available()>0) {

    uint8\_t c = pxSerial.read();

#else

  while(MavSerial.available()>0) {

   uint8\_t c = MavSerial.read();

#endif

    // Try to get a new message

    if(mavlink\_parse\_char(MAVLINK\_COMM\_0, c, &msg, &status)) {

      // Handle message

      switch(msg.msgid) {

        case MAVLINK\_MSG\_ID\_HEARTBEAT:  // #0: Heartbeat

          {

            // E.g. read GCS heartbeat and go into

            // comm lost mode if timer times out

#ifdef SOFT\_SERIAL\_DEBUGGING

            //mySerial.println("PX HB");

#endif

          }

          break;

        case MAVLINK\_MSG\_ID\_SYS\_STATUS:  // #1: SYS\_STATUS

          {

            /\* Message decoding: PRIMITIVE

             \*    mavlink\_msg\_sys\_status\_decode(const mavlink\_message\_t\* msg, mavlink\_sys\_status\_t\* sys\_status)

             \*/

            //mavlink\_message\_t\* msg;

            mavlink\_sys\_status\_t sys\_status;

            mavlink\_msg\_sys\_status\_decode(&msg, &sys\_status);

#ifdef SOFT\_SERIAL\_DEBUGGING

            mySerial.print("PX SYS STATUS: ");

            mySerial.print("[Bat (V): ");

            mySerial.print(sys\_status.voltage\_battery);

            mySerial.print("], [Bat (A): ");

            mySerial.print(sys\_status.current\_battery);

            mySerial.print("], [Comms loss (%): ");

            mySerial.print(sys\_status.drop\_rate\_comm);

            mySerial.println("]");

#else

            Serial.print("PX SYS STATUS: ");

            Serial.print("[Bat (V): ");

            Serial.print(sys\_status.voltage\_battery);

            Serial.print("], [Bat (A): ");

            Serial.print(sys\_status.current\_battery);

            Serial.print("], [Comms loss (%): ");

            Serial.print(sys\_status.drop\_rate\_comm);

            Serial.println("]");

#endif

          }

          break;

        case MAVLINK\_MSG\_ID\_PARAM\_VALUE:  // #22: PARAM\_VALUE

          {

            /\* Message decoding: PRIMITIVE

             \*    mavlink\_msg\_param\_value\_decode(const mavlink\_message\_t\* msg, mavlink\_param\_value\_t\* param\_value)

             \*/

            //mavlink\_message\_t\* msg;

            mavlink\_param\_value\_t param\_value;

            mavlink\_msg\_param\_value\_decode(&msg, &param\_value);

#ifdef SOFT\_SERIAL\_DEBUGGING

            mySerial.println("PX PARAM\_VALUE");

            mySerial.println(param\_value.param\_value);

            mySerial.println(param\_value.param\_count);

            mySerial.println(param\_value.param\_index);

            mySerial.println(param\_value.param\_id);

            mySerial.println(param\_value.param\_type);

            mySerial.println("------ Fin -------");

#endif

          }

          break;

        case MAVLINK\_MSG\_ID\_RAW\_IMU:  // #27: RAW\_IMU

          {

            /\* Message decoding: PRIMITIVE

             \*    static inline void mavlink\_msg\_raw\_imu\_decode(const mavlink\_message\_t\* msg, mavlink\_raw\_imu\_t\* raw\_imu)

             \*/

            mavlink\_raw\_imu\_t raw\_imu;

            mavlink\_msg\_raw\_imu\_decode(&msg, &raw\_imu);

#ifdef SOFT\_SERIAL\_DEBUGGING

            //mySerial.println("PX RAW IMU");

            //mySerial.println(raw\_imu.xacc);

#endif

          }

          break;

        case MAVLINK\_MSG\_ID\_ATTITUDE:  // #30

          {

            /\* Message decoding: PRIMITIVE

             \*    mavlink\_msg\_attitude\_decode(const mavlink\_message\_t\* msg, mavlink\_attitude\_t\* attitude)

             \*/

            mavlink\_attitude\_t attitude;

            mavlink\_msg\_attitude\_decode(&msg, &attitude);

#ifdef SOFT\_SERIAL\_DEBUGGING

            mySerial.println("PX ATTITUDE");

            mySerial.println(attitude.roll);

#else

          Serial.println("PX ROLL");

          Serial.println(attitude.roll);

          Serial.println("PX YAW");

          Serial.println(attitude.yaw);

#endif

          }

          break;

       default:

#ifdef SOFT\_SERIAL\_DEBUGGING

          mySerial.print("--- Otros: ");

          mySerial.print("[ID: ");

          mySerial.print(msg.msgid);

          mySerial.print("], [seq: ");

          mySerial.print(msg.seq);

          mySerial.println("]");

#endif

          break;

      }

    }

  }

}