Altis v4 competition Logger interface to Spektrum TM1000 telemetry system

by Raymond Domingo

(raymond@domingo.nl)

Version 1.0.1, revision 2.0, last update 2014 june 36

Revision history	
2014-06-26	Added sourcode to document addendum

Acknowledgements

A word of thanks to Lukáš Palkovič of AerobTec, s.r.o. who supported me in my quest to interface the Altis v4 logger to the Spektrum TM1000 telemetry x-bus with required information and prototype hardware. And to those people who reverse engineered the Spektrum x-bus protocol, especially Mukenukem, without this project wouldn't be possible.

Disclaimer

Information in this document is provided as is, there is no warranty or what so ever.
Information in this document describes an experimental prototype, it's assumed to contain bugs and even might cause wide system failures. Usage is at your own risk, the author can't be held responsible.

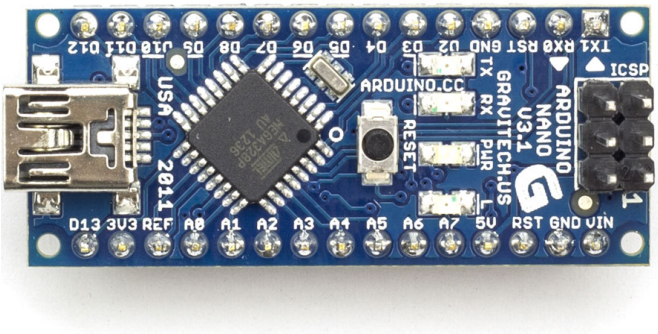
Introduction

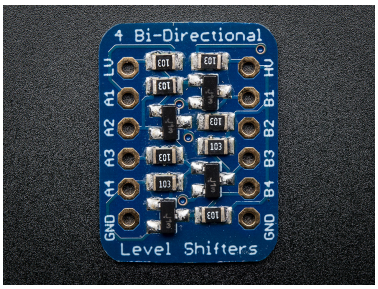
This document describes how to build an interface between the Altis v4 logger and the Spektrum TM1000 telemetry (x-bus) system using Arduino technology.

Requirements

- The Application SourceCode: altisSpektrumInterfaceV1.ino
- Arduino nano V3 + USB Cable, might also work with other arduino variants supporting I2C and serial communication.
- Level shifter, for example:
  4-channel I2C-safe Bi-directional Logic Level Converter - BSS138
- X-bus connector
- Altis v4 com port connector
- Connector compliant with your power supply,
  For arduino nano V3 is the recommended Input Voltage 7-12 V
- Lots of cables
- *Optional: hotglue to keep all in place and protect vulnerable components.*

Hardware setup





Wire the spektrum TM1000 to the Level shifter and connect level shifter to the Arduino as described in the table below:

TM1000	**LEVEL SHIFTER (LV)**	**LEVEL SHIFTER (HV)**	**ARDUINO**
SDA	A1	B1	A4
SCL	A2	B2	A5
5V **might be 3.?V**	LV	HV	5V
GND	GND (LV side)	GND	GND

Configure the COM B of the Altis Logger for “Live_Output”

ALTIS V4 LOGGER COM B	**ARDUINO**
orange / pgd com	RX0
red (3,5 - 7v)	
brown / gnd	GND

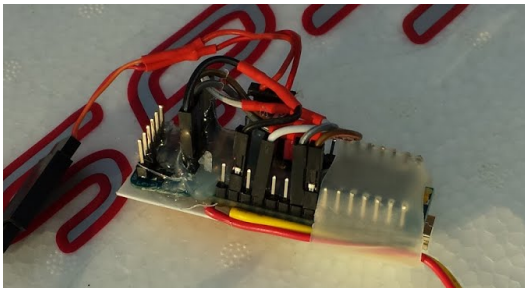
Software

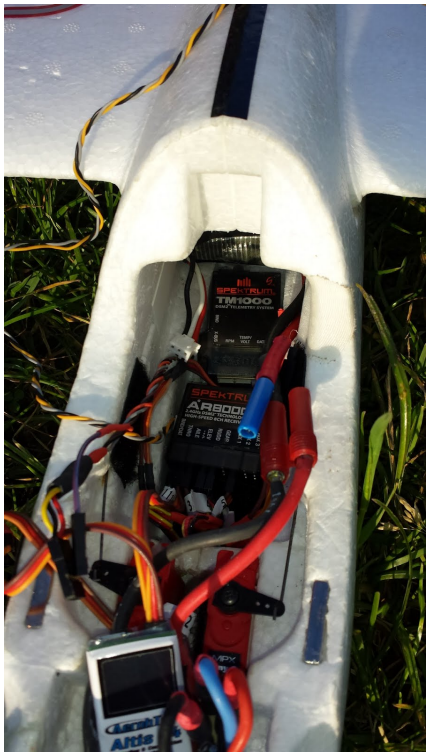
The source code of the program can be found in the addendum: Source code

Test

I used described hardware / software once, during a few minutes test flight in my multiplex easyglider pro test configuration. See also this short demo movie: http://youtu.be/tyjLrOhxg-A

Photo impression:





Addendum: Source code

```
#include <Wire.h>
#include <swI2C_BMP085.h>
#include <I2cMaster.h>

////////////////////////////////////////////////////////
// Altis V4 competition logger interface to
// Spektrum TM1000 telemetry system using x-bus
//
// Version 1.0, 2014 june 2
// By Raymond Domingo, code is inspired by several examples found on the web
// Special thanks to Lukáš Palkovič of AerobTec, s.r.o who supported the project
// wherever possible.
//
////////////////////////////////////////////////////////


// Data string 1.I2C adress,second byte 0x00, then MSB, LSB

//I2C Adress out Byte 0, I2C adress for altimeter
byte tmpSpektrumData1[] = {0x12, 0x00, 0x00, 0x00, 0x00, 0x0, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00};
byte i2cAdress1         = tmpSpektrumData1[0];

//I2C Adress out Byte 0, I2C adress for g-force
byte tmpSpektrumData2[] = {0x14, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00};
byte i2cAdress2         = tmpSpektrumData2[0];

int16_t hmax;

// VARIABLE TABLE
// ==============
// 0 - Global time
// 1 - altitude
// 2 - vario
// 3 - F5J height
// 4 - pressure
// 5 - temperature
// 6 - voltage
// 7 - free memory
// 8 - throttle input
```

```
// 9 - throttle output
// 10 - firmware version
// 11 - serial number
char* labels[] = {
  "Global time  ",
  "Altitude       ",
  "Vario          ",
  "F5J height   ",
  "Presure      ",
  "Temperature          ",
  "Voltage      ",
  "Free memory          ",
  "Throttle input   ",
  "Throttle output  ",
  "Firmware version ",
  "Serial number        "
};
String globalTime;
String altitude;
String vario;
String f5jHeight;
String pressure2;
String temperature;
String voltage;
String freeMemory;
String throttleInput;
String throttleOutput;
String fwVersion;
String serialNumber = "";
// Index of variable currently read (zee the variable table above)
int variableIndex = 0;

//int testOffset = 0;
// used to read a value from altis logger char by char
String valueBuffer = "";

boolean testMode = false;
int testOutPin = 2;
int testInPin = 3;
int testLedPin = 13;


void setup() {
```

```
  ////////////////////////////////////////////////////
  // SPEKTRUM TELEMETRY COMMUNICATION INITIALIZATION
  ////////////////////////////////////////////////////
  Wire.begin(i2cAdress1);
  uint8_t SlaveAddress1 = (i2cAdress1 << 1); // TWI_ADR_BITS=1 shift the adress one position
to left.
  uint8_t SlaveAddress2 = (i2cAdress2 << 1); // The second slave address to respond to.
  // XOR the addresses to get the address mask and store it in the TWAR register. The Arduino
respond now to the adress 0x14 and 0x12.
  TWAMR = SlaveAddress1 ^ SlaveAddress2; // Initialize I2C lib & setup slave's I2C address.
  Wire.onRequest(requestEvent);//Register the function requestData to be called when the
TM1000 requests data from this slave device.

  ////////////////////////////////////////////////////
  // ALTIS LOGGER COMMUNICTION INITIALIZATION
  ////////////////////////////////////////////////////
  //Initialize serial and wait for port to open.
  //Altislogger seems to require this baud rate
  Serial.begin(115200);
  while (!Serial) {
        ; // wait for serial port to connect. Needed for Leonardo only
  }
  pinMode(testLedPin, OUTPUT);
  pinMode(testOutPin, OUTPUT);
  pinMode(testInPin, INPUT);
  digitalWrite(testOutPin, HIGH);
}

void loop() {
  // remember data changes to determine when we need to send telemetry data
  boolean dataChanged = false;

  // check if data is available
  if (Serial.available() > 0) {

        // read byte
        int thisByte = Serial.read();
        // convert byte to character
        char c = (char)thisByte;

        if (c == ',' || thisByte == 13) {
        // found control character, indicating variable completely read
        // Logging
```

```
        // String label = labels[variableIndex];
        // Serial.println(label + " = " + valueBuffer);

        if (variableIndex == 0) {
        globalTime = valueBuffer;
        } else if (variableIndex == 1) {
        if (valueBuffer != altitude) {
        dataChanged = true;
        }
        altitude = valueBuffer;
        } else if (variableIndex == 2) {
        vario = valueBuffer;
        } else if (variableIndex == 3) {
        f5jHeight = valueBuffer;
        } else if (variableIndex == 4) {
        pressure2 = valueBuffer;
        } else if (variableIndex == 5) {
        temperature = valueBuffer;
        } else if (variableIndex == 6) {
        voltage = valueBuffer;
        } else if (variableIndex == 7) {
        freeMemory = valueBuffer;
        } else if (variableIndex == 8) {
        throttleInput = valueBuffer;
        } else if (variableIndex == 9) {
        throttleOutput = valueBuffer;
        } else if (variableIndex == 10) {
        fwVersion = valueBuffer;
        } else if (variableIndex == 11) {
        if (serialNumber != "" && serialNumber != valueBuffer) {
        // Serial.println("Data corruption, serial is:" + serialNumber + ", valueBuffer:" +
valueBuffer);
        } else {
        serialNumber = valueBuffer;
        }
        }

        // reset buffer
        valueBuffer = "";
        // update index
        variableIndex++;

        } else if (thisByte == 10) {
```

```
        // end of line detected, prepare for next record
        variableIndex = 0;
        valueBuffer = "";

        } else {
        // no special character, append character to valueBuffer
        valueBuffer = valueBuffer + c;
        }
 }

  // TESTING MODE
//  int testPinState = digitalRead(testInPin);
//  Serial.println("testPinState:");
//  Serial.println(testPinState);
//  if (testPinState == HIGH) {
//        // enter testing mode
//        testMode = true;
//        // led will indicate we are testing
//        digitalWrite(testLedPin, HIGH);
//        // write testvalue to f5jHeight field
//        altitude = "123";
//        dataChanged = true;
//  } else if (testMode) {
//        // leaf testing mode
//        testMode = false;
//        digitalWrite(testLedPin, LOW);
//        // reset data if needed
//        // (f5jHeight might already contain new data
//        if (altitude = "123") {
//        altitude = "0";
//        dataChanged = true;
//        }
//  }

  if (dataChanged) {
        // data interesting for spektrum tm1000 is changed,
        // send latest data;
        sendSpektrumData();
  }
}


void sendSpektrumData()
```

```
{
  int hi = altitude.toInt()*10;
//  Serial.println("-------------------------");
//  Serial.println("altitude (substring_1):" + altitude.substring(1));
//  Serial.print("hi                    :");
//  Serial.println(hi);
//  Serial.println("-------------------------");
  delay(10);

  byte SpektrumData_2;//Altitude MSB
  byte SpektrumData_3;//Altitude LSB
  byte SpektrumData_4;//Max Altitude MSB
  byte SpektrumData_5;//Max Altitude LSB

  SpektrumData_2 = highByte(hi); //2
  SpektrumData_3 = lowByte(hi); //3

  if (hi > hmax)//Bestimmt die maximale Höhe -  measures the max hight
  {
          hmax = hi;
          SpektrumData_4 = highByte(hmax); //4
          SpektrumData_5 = lowByte(hmax); //5
  }

  tmpSpektrumData1[2] =  SpektrumData_2; //Schreibt MSB Höhe; MSB Altitude
  tmpSpektrumData1[3] =  SpektrumData_3; //Schreibt LSB Höhe; LSB Altitude

  tmpSpektrumData1[4] =  SpektrumData_4; //Schreibt MSB Max Höhe
  tmpSpektrumData1[5] =  SpektrumData_5;//Schreibt LSB Max Höhe

  tmpSpektrumData2[2] =  highByte(hi); //2 MSB Höhe dm.cm;  altitude dm.cm
  tmpSpektrumData2[3] =  lowByte(hi); //3 LSB Höhe dmcm; altitude cm

  int iVario = vario.toInt();
  tmpSpektrumData2[6] =  highByte(iVario); //2 MSB Vario to MSB z
  tmpSpektrumData2[7] =  lowByte(iVario); //3 LSB Vario to LSB z
}

void requestEvent()//Called after the TM1000 requests data from this slave device.
{
  uint8_t Called_Adress = (TWDR >> 1); // The called adress from the I2C master (TM1000)
  //digitalWrite(redLEDPin, HIGH); // Shown that an event are catched, it's a test function only
  if (Called_Adress == i2cAdress1)//If x == 0x12 then send tmpSpektrumdata2 from 0x12
```

```
  {
        Wire.write(tmpSpektrumData1, 16);//Wire.write(byteData, length) send the arrary
tmpSpektrumData1 to the TM1000
  }

  if (Called_Adress == i2cAdress2)//If x == 0x14 then send tmpSpektrumdata adress 0x14
  {
        Wire.write(tmpSpektrumData2, 16);//Wire.write(byteData, length) send the arrary
tmpSpektrumData2 to the TM1000
  }
}
```