SBUS

Library for communicating with SBUS receivers and servos using Teensy 3.x and Teensy LC devices.

Description

SBUS is a protocol for RC receivers to send commands to servos. Unlike PWM, SBUS uses a bus architecture where a single signal line can be connected up to 16 servos with each receiving a unique command. SBUS capable servos are required; each can be programmed with a unique address (Channel 0 - 15) using an SBUS servo programmer. Advantages of SBUS include the reduction of wiring clutter and ease of parsing commands from RC receivers.

Much of the information gathered on the SBUS protocol comes from Uwe Gartmann. Implementation of sending SBUS packet data to servos on Teensy devices was greatly aided by this discussion and feedback from Paul Stroffregen.

The SBUS protocol uses inverted serial logic with a baud rate of 100000, 8 data bits, even parity bit, and 2 stop bits. The SBUS packet is 25 bytes long consisting of:

- Byte[0]: SBUS Header, 0x0F
- Byte[1-22]: 16 servo channels, 11 bits per servo channel
- Byte[23]:
  - Bit 7: digital channel 17 (0x80)
  - Bit 6: digital channel 18 (0x40)
  - Bit 5: frame lost (0x20)
  - Bit 4: failsafe activated (0x10)
  - Bit 0 - 3: n/a
- Byte[24]: SBUS End Byte, 0x00

A table mapping bytes[1-22] to servo channels is included.

This library has two basic modes of functionality:

1. Reading and parsing SBUS packets. This is useful for using an SBUS capable receiver to receive commands from a RC transmitter and parse the SBUS packet for vehicle control (closed loop control laws, servo mapping, mode change commands) and logging. SBUS packet reading

and parsing is available with raw count data and calibrated to a +/- 1.0 float. Additionally, lost frame and failsafe data is made available.
2. Writing SBUS packets. This is useful for commanding up to 16 SBUS capable servos from the Teensy device.

This library has been tested using FrSky SBUS capable receivers (X8R and X4R) and FrSky SBUS capable servos (D25MA). Feedback from users, especially with other brand receivers and servos (i.e. Futaba), would be greatly appreciated.

Usage

This library uses the hardware serial for Teensy devices. Additionally, this library **requires Teensyduino 1.30 or above**.

Simply clone or download and extract the zipped library into your Arduino/libraries folder.

Bind your SBUS capable receiver to your transmitter. Setup your SBUS capable servos by programming each with a unique channel number.

SBUS(HardwareSerial& bus) A SBUS object should be declared, specifying the hardware serial port the SBUS receiver and servos are connected to. For example, the following code declares a SBUS object called *x8r* located on the Teensy hardware serial port 1:

```
SBUS x8r(Serial1);
```

void begin() This should be called in your setup function. It initializes the serial communication between the Teensy and SBUS receiver and servos.

```
x8r.begin();
```

bool read(uint16_t* channels, uint8_t* failsafe, uint16_t* lostFrames) *read(uint16_t* channels, uint8_t* failsafe, uint16_t* lostFrames)* reads data from the SBUS receiver and parses the SBUS packet. When a complete packet is received, *read(uint16_t* channels, uint8_t* failsafe, uint16_t* lostFrames)* returns *true* and the *channels[0-15]*, *failsafe*, and *lost frames*data is available. Note that *lost frames* is a counter that increments once each time a lost frame flag is read in the SBUS packet. For example, placing the following code in the loop function will print the value of *channel 0* every time a valid SBUS packet is received.

```
uint16_t channels[16];
uint8_t failSafe;
uint16_t lostFrames = 0;
```

```
if(x8r.read(&channels[0], &failSafe, &lostFrames)){
        Serial.println(channels[0]);
}
```

bool readCal(float* calChannels, uint8_t* failsafe, uint16_t* lostFrames) *readCal(float* calChannels, uint8_t* failsafe, uint16_t* lostFrames)* reads data from the SBUS receiver and parses the SBUS packet. The data from *channels[0-15]* is calibrated to a +/- 1.0 float value assuming a linear relationship based on the minimum and maximum value (172 and 1811 using FrSky set to 0-100%). When a complete packet is received, *readCal(float* calChannels, uint8_t* failsafe, uint16_t* lostFrames)* returns *true* and the *calChannels[0-15]*, *failsafe*, and *lost frames* data is available. Note that *lost frames* is a counter that increments once each time a lost frame flag is read in the SBUS packet. For example, placing the following code in the loop function will print the calibrated value of *channel 0* every time a valid SBUS packet is received.

```
float channels[16];
uint8_t failSafe;
uint16_t lostFrames = 0;

if(x8r.readCal(&channels[0], &failSafe, &lostFrames)){
        Serial.println(channels[0]);
}
```

void write(uint16_t* channels) *write(uint16_t* channels)* writes the SBUS packet to SBUS capable servos given position commands from *channels[0-15]*. Note that this function simply creates and sends the SBUS packet, but does not handle timing (i.e. the time between sending subsequent SBUS packets). This timing must be handled by the calling function. For example, placing the following code in the loop function will create and send the SBUS packet to servos every time a valid SBUS packet is received.

```
uint16_t channels[16];
uint8_t failSafe;
uint16_t lostFrames = 0;

// look for a good SBUS packet from the receiver
if(x8r.read(&channels[0], &failSafe, &lostFrames)){
        // write the SBUS packet to SBUS compatible servos
    x8r.write(&channels[0]);
}
```

Wiring

Please refer to the Teensy pinout diagrams for hardware serial port pin information. The SBUS capable receiver ground should be connected to the Teensy ground. The receiver power may be connected to the Teensy Vin or to an external 5V power source. Receiver signal should be connected to the

Teensy RX pin on the specified hardware serial port. The SBUS capable servo ground should be connected to the Teensy ground. **Servo power must come from an external power source**; Vin is not likely capable of supplying the necessary current. Servo signal should be connected to the Teensy TX pin on the specified hardware serial port.