ThinSat Program TSLPB Library

0.6.0

Generated by Doxygen 1.8.14

Contents

Chapter 1

Twiggs Space Lab Payload Board Driver

TSLPB is a driver class that can be instantiated and used to access the sensors and devices on the TSLPB V3 for the ThinSat program.

The driver sets up all the input and output pins required for accessing the analog sensors, and provides methods for reading both the analog and digital sensors.

1.1 Basic Usage

You will need to do the following to use this library:

1. Include TSLPB.h in your program.
2. Instantiate a TSLPB object
3. Run the TSLPB::begin() method

Once these steps are complete, you may call any of the public methods to interact with the TSL Payload Board.

1.2 Installation

Installing the library is easy using the Arduino IDE, which can be downloaded at https://www.arduino.↩cc/en/Main/Software

Once you have the Arduino IDE installed, use the menu and navigate to Sketch > Include Library > Manage Libraries...

This opens the Library Manager. Type "thinsat" into the search bar. Select the library, and click "Install"

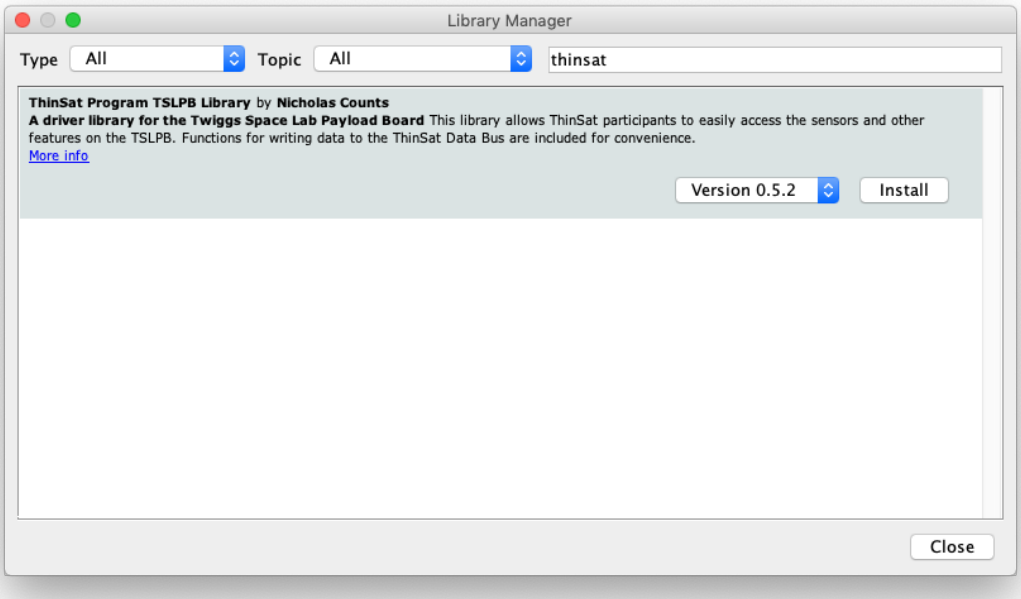

Figure 1.1 Library Manager

1.2.1 Manual Installation

You can always download the latest version of the library as a zip file from https://github.com/VCSFA-MARS/TSLPB/releases/latest

Then use the Sketch > Include Library > Add Zip Library... feature of the Arduino IDE to install the library.

1.3 Getting Started

The TSLPB Library includes several sample sketches to help you get coding and illustrate some of the features.

- EEPROM - read and write to the onboard memory chip
- i2c_scanner - find all I2C devices connected to the TSLPB
- serial_plot - example of reading the accelerometer with live ouput plot
- simple - a blank sketch with the library includes set up.
- template - a good starting point for developing your code
- VCSFA_ThinSat - an example of the VCSFA ThinSat flight software

Open the serial_plot example in the Arduino IDE. File > Examples > ThinSat Program TSLPB Library > serial_plot

Plug your programming cable into the TSLPB diagnostic connector. Set the Arduino IDE port to use the programming cable with Tools > Port. Click the "Upload" button, which looks like an arrow pointing right (->)

Figure 1.2 Sketch compiled and uploaded

The example sketch will compile and upload to the board. When the upload completes, open the Serial Plotter (Tools > Serial Plotter) and set the baud rate to 9600, which is the default baud rate for the TSLPB diagnostic port.

The Serial Plotter will launch and begin graphing the values of the onboard gyroscope. Try moving the board around and watch how the plot changes.

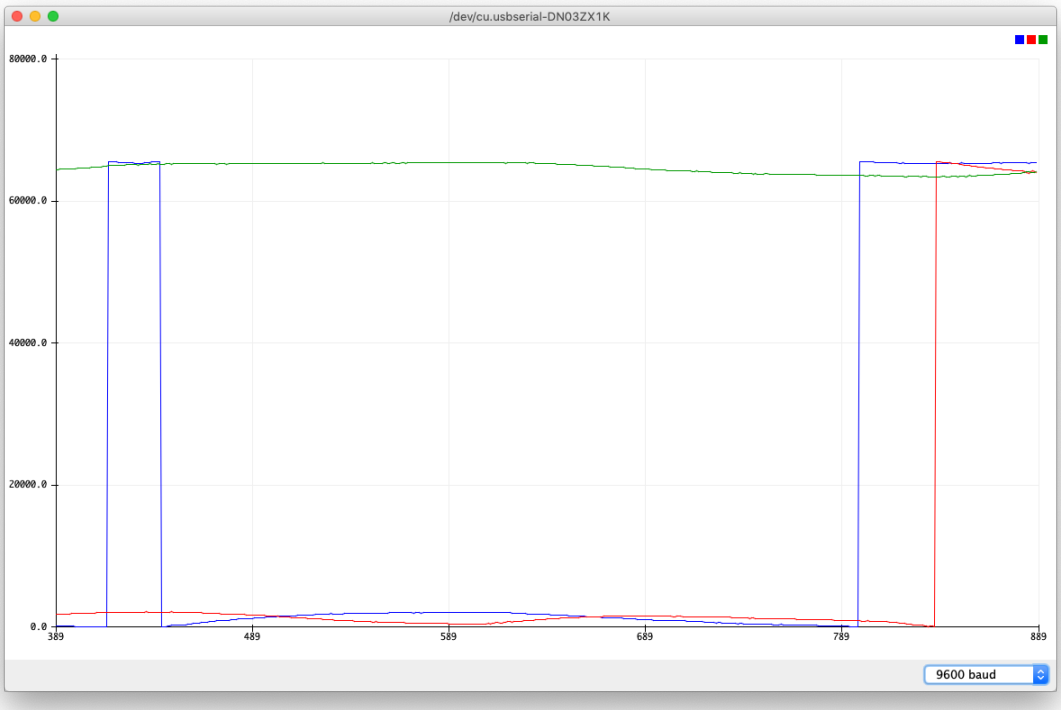

Figure 1.3 Serial Plotter with Gyroscope data

1.3.1 Example

```
#include "TSLPB.h"

TSLBP tslpb;
ThinsatPacket_t missionData;

void setup() {
    tslpb.begin();
}

void loop() {
    uint16_t tslVolts   = tslpb.readAnalogSensor(Voltage);
    uint16_t tslCurrent = tslpb.readAnalogSensor(Current);
    uint16_t tslTempExt = tslpb.readAnalogSensor(TempExt);

    uint16_t tslDT1Raw  = tslpb.readTSLDigitalSensorRaw(DT1);
    double   tslDT1C    = tslpb.readTSLDigitalSensor(DT1);

    missionData.payloadData.solar = tslpb.readAnalogSensor(Solar);

    while (!tslpb.isClearToSend())
    {
        delay(100);
    }

    tslpb.pushDataToNSL(missionData);
}
```

You probably noticed the "Voltage", "Current", etc arguments. The TSLPB driver has two enums that allow the client to call the read methods with human-readable code, and without worrying about keeping I2C addresses or managing low-level mux switching.

- TSLPB_AnalogSensor_t
- TSLPB_DigitalSensor_t

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 ThinsatPacket_t Union Reference

A union of the UserDataStruct_t payloadData and a byte array that is used to send the user's mission data to the NSL Mothership.

`#include <ThinSat_DataPacket_generic.h>`

Public Attributes

- UserDataStruct_t **payloadData**
- byte **NSLPacket** [sizeof(UserDataStruct_t)]

4.1.1 Detailed Description

A union of the UserDataStruct_t payloadData and a byte array that is used to send the user's mission data to the NSL Mothership.

Warning

> DO NOT MODIFY THIS UNION UNLESS YOU REALLY REALLY KNOW WHAT YOU ARE DOING. This datatype is used in the public method TSLPB::pushDataToNSL(ThinsatPacket_t data) and changing this union may break that functionality.

The documentation for this union was generated from the following file:

- ThinSat_DataPacket_generic.h

4.2 TSLPB Class Reference

The controller class for the TSL Payload Board. Create an instance of this class to use its member functions for accessing the onboard analog and digital sensors. Methods for communicating with the NSL Mothership are also included.

`#include <TSLPB.h>`

Public Member Functions

- void begin ()

  Initializes the TSLPB, starts the I2C bus, and configures the pins needed for reading the TSLPB analog sensors.
- bool pushDataToNSL (ThinsatPacket_t data)

  This function sends the user's payload data to NSL Mothership over the serial line. This function expects a ThinsatPacket_t union as an argument. That data type is defined in ThinSat_DataPacket.h, and the contents of the user data structure may be customized.
- uint16_t readAnalogSensor (TSLPB_AnalogSensor_t sensorName)

  This method returns the raw value from the specified analog sensor.
- double readDigitalSensor (TSLPB_DigitalSensor_t sensor)

  This API returns the process from the specified sensor as a double-precision floating point value in the appropriate units for the sensor.
- uint16_t readDigitalSensorRaw (TSLPB_DigitalSensor_t sensor)

  This API returns the raw value from the specified sensor. Handles endiannes and discarding unused bits.
- uint8_t getMemByte (uint16_t reg)

  This function reads data from the TSLPB EEPROM chip. This is used to retrieve data from nonvolatile storage. One byte is read from the specified register.
- void putMemByte (uint16_t reg, uint8_t data)

  This function writes data to the TSLPB EEPROM chip. This is used to store data in nonvolatile storage to allow persistance in the event of power loss. One byte is written to the specified register.
- template<class TYPE >
  void readMemVar (word reg, TYPE &result)

  This function reads data from the TSLPB EEPROM chip. This is used to retrieve data from nonvolatile storage. The data argument will be filled with data from the EEPROM interpreted as data's typedef.
- template<class TYPE >
  void writeMemVar (word reg, TYPE varToWrite)

  This function writes data to the TSLPB EEPROM chip. This is used to store data in nonvolatile storage to allow persistance in the event of power loss. The entire contents of the data argument will be written to the EEPROM.
- void sleepUntilClearToSend ()

  This function places the TSLPB into a low power sleep mode until the NSL "Mothership" signals that it is ready to receive data on the NSL Bus.
- bool isClearToSend ()

  This function returns true if the NSL Mothership is ready to receive data over the serial line.

Public Attributes

- SoftwareSerial NSLbus

  NSL Software Serial bus object.
- bool isMagnetometerOverflow = false

  Overflow status of magnetometer registers.

4.2.1 Detailed Description

The controller class for the TSL Payload Board. Create an instance of this class to use its member functions for accessing the onboard analog and digital sensors. Methods for communicating with the NSL Mothership are also included.

4.2.2 Member Function Documentation

4.2.2.1 begin()

```
void TSLPB::begin ( )
```

Initializes the TSLPB, starts the I2C bus, and configures the pins needed for reading the TSLPB analog sensors.

Call this function in the setup() function as follows:

```
void setup() {
    tslpb.begin();
}
```

Note

This function changes the state of 4 I/O pins:

PIN	MODE
TSL_ADC	Analog Input
TSL_MUX_A	Digital Output
TSL_MUX_B	Digital Output
TSL_MUX_C	Digital Output
TSL_NSL_BUS_STATUS_PIN	Digital Input

4.2.2.2 getMemByte()

```
uint8_t TSLPB::getMemByte (
            uint16_t reg )
```

This function reads data from the TSLPB EEPROM chip. This is used to retrieve data from nonvolatile storage. One byte is read from the specified register.

Parameters

in	*reg*	a two-byte (word) unsigned integer

Returns

data a single byte. May be signed, unsigned, char, etc.

4.2.2.3 isClearToSend()

```
bool TSLPB::isClearToSend ( )
```

This function returns true if the NSL Mothership is ready to receive data over the serial line.

Returns

true or false

4.2.2.4 pushDataToNSL()

```
bool TSLPB::pushDataToNSL (
            ThinsatPacket_t data )
```

This function sends the user's payload data to NSL Mothership over the serial line. This function expects a ThinsatPacket_t union as an argument. That data type is defined in ThinSat_DataPacket.h, and the contents of the user data structure may be customized.

Parameters

in	*data*	A ThinsatPacket_t union.
out	*bool*	Successfull transmission status

Returns

nominal transmission: true or false

4.2.2.5 putMemByte()

```
void TSLPB::putMemByte (
            uint16_t reg,
            uint8_t data )
```

This function writes data to the TSLPB EEPROM chip. This is used to store data in nonvolatile storage to allow persistance in the event of power loss. One byte is written to the specified register.

Parameters

in	*reg*	a two-byte (word) unsigned integer
in	*data*	a single byte. May be signed, unsigned, char, etc.

Note

This method will print an error message to the diagnostic serial port if the write fails.

4.2.2.6 readAnalogSensor()

```
uint16_t TSLPB::readAnalogSensor (
            TSLPB_AnalogSensor_t sensorName )
```

This method returns the raw value from the specified analog sensor.

Parameters

in	*sensorName*	: TSLPB_AnalogSensor_t Sensor Name enum

Returns

a uint16_t containing raw value of the Arduino Pro Mini's ADC.

Note

The TSLPB uses a 10-bit Analog-to-Digital Converter. The 6 MSbs of the return value will always be 0.

4.2.2.7 readDigitalSensor()

```
double TSLPB::readDigitalSensor (
            TSLPB_DigitalSensor_t sensorName )
```

This API returns the process from the specified sensor as a double-precision floating point value in the appropriate units for the sensor.

Parameters

in	*sensorName*	TSLPB_DigitalSensor_t Sensor Name Selection Enum

Returns

a value in the appropriate units for the sensor as a double precision floating point value.

4.2.2.8 readDigitalSensorRaw()

```
uint16_t TSLPB::readDigitalSensorRaw (
            TSLPB_DigitalSensor_t sensorName )
```

This API returns the raw value from the specified sensor. Handles endiannes and discarding unused bits.

Parameters

in	*sensorName*	: TSLPB_DigitalSensor_t Sensor Name Enum

Returns

a uint16_t containing the bit pattern from the sensor's register.

< I2C buffer for read function

< return value, after endian correction

4.2.2.9 readMemVar()

```
template<class TYPE >
void TSLPB::readMemVar (
            word reg,
            TYPE & result )  [inline]
```

This function reads data from the TSLPB EEPROM chip. This is used to retrieve data from nonvolatile storage. The data argument will be filled with data from the EEPROM interpreted as data's typedef.

Parameters

in	*reg*	a two-byte (word) unsigned integer
out	*data*	any type is allowed. assigned a value by readMemVar()

Note

This method will handle any data type and write the appropriate number of registers on the EEPROM. Care must be taken to ensure you start at the correct register.

4.2.2.10 sleepUntilClearToSend()

```
void TSLPB::sleepUntilClearToSend ( )
```

This function places the TSLPB into a low power sleep mode until the NSL "Mothership" signals that it is ready to receive data on the NSL Bus.

Warning

This function is not implemented

4.2.2.11 writeMemVar()

```
template<class TYPE >
void TSLPB::writeMemVar (
            word reg,
            TYPE varToWrite )  [inline]
```

This function writes data to the TSLPB EEPROM chip. This is used to store data in nonvolatile storage to allow persistance in the event of power loss. The entire contents of the data argument will be written to the EEPROM.

Parameters

in	*reg*	a two-byte (word) unsigned integer
in	*data*	any type is allowed.

Note

This method will handle any data type and write the appropriate number of registers on the EEPROM. Care must be taken to ensure you do not overwrite existing data.

The documentation for this class was generated from the following files:

- TSLPB.h
- TSLPB.cpp

4.3 UserDataStruct_t Struct Reference

A generic data structure to hold any data the user intends to send back to Earth.

`#include <ThinSat_DataPacket_generic.h>`

Public Attributes

- char **header** [NSL_PACKET_HEADER_LENGTH]
- int8_t b1

  b1 (Generic packet byte 1 of 35)
- int8_t b2

  b2 (Generic packet byte 2 of 35)
- int8_t b3

  b3 (Generic packet byte 3 of 35)
- int8_t b4

  b4 (Generic packet byte 4 of 35)
- int8_t b5

  b5 (Generic packet byte 5 of 35)
- int8_t b6

  b6 (Generic packet byte 6 of 35)
- int8_t b7

  b7 (Generic packet byte 7 of 35)
- int8_t b8

  b8 (Generic packet byte 8 of 35)
- int8_t b9

  b9 (Generic packet byte 9 of 35)
- int8_t b10

  b10 (Generic packet byte 10 of 35)
- int8_t b11

  b11 (Generic packet byte 11 of 35)
- int8_t b12

  b12 (Generic packet byte 12 of 35)
- int8_t b13

  b13 (Generic packet byte 13 of 35)
- int8_t b14

  b14 (Generic packet byte 14 of 35)
- int8_t b15

  b15 (Generic packet byte 15 of 35)

- int8_t b16

  b16 (Generic packet byte 16 of 35)
- int8_t b17

  b17 (Generic packet byte 17 of 35)
- int8_t b18

  b18 (Generic packet byte 18 of 35)
- int8_t b19

  b19 (Generic packet byte 19 of 35)
- int8_t b20

  b20 (Generic packet byte 20 of 35)
- int8_t b21

  b21 (Generic packet byte 21 of 35)
- int8_t b22

  b22 (Generic packet byte 22 of 35)
- int8_t b23

  b23 (Generic packet byte 23 of 35)
- int8_t b24

  b24 (Generic packet byte 24 of 35)
- int8_t b25

  b25 (Generic packet byte 25 of 35)
- int8_t b26

  b26 (Generic packet byte 26 of 35)
- int8_t b27

  b27 (Generic packet byte 27 of 35)
- int8_t b28

  b28 (Generic packet byte 28 of 35)
- int8_t b29

  b29 (Generic packet byte 29 of 35)
- int8_t b30

  b30 (Generic packet byte 30 of 35)
- int8_t b31

  b31 (Generic packet byte 31 of 35)
- int8_t b32

  b32 (Generic packet byte 32 of 35)
- int8_t b33

  b33 (Generic packet byte 33 of 35)
- int8_t b34

  b34 (Generic packet byte 34 of 35)
- int8_t b35

  b35 (Generic packet byte 35 of 35)
- int16_t quatw

  1 - 2 (value from -4000 to 4000) 4.000 (unitless)
- int16_t quatx

  3 - 4 (value from -1000 to 1000) 1.000 (unitless)
- int16_t quaty

  5 - 6 (value from -1000 to 1000) 1.000 (unitless)
- int16_t quatz

  7 - 8 (value from -1000 to 1000) 1.000 (unitless)
- int16_t bnomagx

  9 - 10 (value from -20480 to 20470) 2047.0 uT (from BNO)
- int16_t bnomagy

11 - 12 (value from -20480 to 20470) 2047.0 uT

- int16_t bnomagz

  13 - 14 (value from -20480 to 20470) 2047.0 uT
- uint8_t bnoCal

  15 (sys, gyro, accel, mag) 01010101b
- unsigned long bmePres

  16 - 19 (values from 0 to 1010000) 101000.0 Pa
- int16_t bmeTemp

  20 - 21 (values from -1000 to 1000) 100.0 C
- uint16_t tslTempExt

  22 - 23 (10 bits 0-1023) ADC Raw Counts
- uint16_t tslVolts

  24 - 25 (10 bits 0-1023) ADC Raw Counts
- uint16_t tslCurrent

  26 - 27 (10 bits 0-1023) ADC Raw Counts
- int16_t tslMagXraw

  28 - 29 Raw value (2's compliment form) -0x7FF8 to 0x7FF8
- int16_t tslMagYraw

  30 - 31 Raw value (2's compliment form) -0x7FF8 to 0x7FF8
- int16_t tslMagZraw

  32 - 33 Raw value (2's compliment form) -0x7FF8 to 0x7FF8
- uint16_t solar

  34 - 35 (10 bits 0-1023) ADC Raw Counts

4.3.1 Detailed Description

A generic data structure to hold any data the user intends to send back to Earth.

A user-customizable structure to hold any data the user intends to send back to Earth.

Note

This is a sample UserDataStruct_t. It was developed to be used by ThinSat participants who do not want to make their own UserDataStruct_t. Each field is simply the byte position (1-based) from 1 to 35. Users will need to split multi-byte data appropriately and will need to type cast their data when storing it in this structure. We recommend adding comments that show the expected ranges and units of any data being put into a field. This will ensure that you can translate the data later.

Warning

The struct must be NSL_PACKET_SIZE bytes in total size. The first member must always be called "header" and have a size of NSL_PACKET_HEADER_LENGTH

Note

This is a sample UserDataStruct_t. It was developed for the VCSFA ThinSat custom payload. Some of the fields are for external sensors and some of the fields are for TSLPB sensors.
We recommend adding comments that show the expected ranges and units of any data being put into a field. This will ensure that you can translate the data later.

Warning

The struct must be NSL_PACKET_SIZE bytes in total size. The first member must always be called "header" and have a size of NSL_PACKET_HEADER_LENGTH

The documentation for this struct was generated from the following files:

- ThinSat_DataPacket_generic.h
- VCSFA_ThinSat_DataPacket.h

Chapter 5

File Documentation

5.1 MPU9250_REGS.h File Reference

Register map and configuration data for the MPU9250 IMU on the TSLPB V3.

Macros

- #define GYRO_FULL_SCALE_250_DPS 0x00

  Gyroscope range parameter.
- #define GYRO_FULL_SCALE_500_DPS 0x08

  Gyroscope range parameter.
- #define GYRO_FULL_SCALE_1000_DPS 0x10

  Gyroscope range parameter.
- #define GYRO_FULL_SCALE_2000_DPS 0x18

  Gyroscope range parameter.
- #define ACC_FULL_SCALE_2_G 0x00

  Accelerometer range parameter.
- #define ACC_FULL_SCALE_4_G 0x08

  Accelerometer range parameter.
- #define ACC_FULL_SCALE_8_G 0x10

  Accelerometer range parameter.
- #define ACC_FULL_SCALE_16_G 0x18

  Accelerometer range parameter.
- #define MAG_MAX_BYTE_VALUE 0x7FF8

  Max register for scaling.
- #define MAG_MAX_VALUE_FLOAT 4912

  in units of uT for scaling

Enumerations

- enum **MPU9250_TEMP_REGISTER_t** { **MPU9250_TEMP_OUT_MSB** = 0x41, **MPU9250_TEMP_OUT_L↩ SB** = 0x42 }
- enum {
  MPU9250_ACCEL_XOUT_MSB = 0x3B, **MPU9250_ACCEL_XOUT_LSB** = 0x3C, **MPU9250_ACCEL_Y↩ OUT_MSB** = 0x3D, **MPU9250_ACCEL_YOUT_LSB** = 0x3E,
  MPU9250_ACCEL_ZOUT_MSB = 0x3F, **MPU9250_ACCEL_ZOUT_LSB** = 0x40, **MPU9250_ACCEL_SE↩ LF_TEST_X** = 0x0D, **MPU9250_ACCEL_SELF_TEST_Y** = 0x0E,
  MPU9250_ACCEL_SELF_TEST_Z = 0x0F }
- enum **MPU9250_GYRO_REGISTER_t** {
  MPU9250_GYRO_XOUT_MSB = 0x43, **MPU9250_GYRO_XOUT_LSB** = 0x44, **MPU9250_GYRO_YOUT↩ _MSB** = 0x45, **MPU9250_GYRO_YOUT_LSB** = 0x46,
  MPU9250_GYRO_ZOUT_MSB = 0x47, **MPU9250_GYRO_ZOUT_LSB** = 0x48, **MPU9250_GYRO_SELF↩ _TEST_X** = 0x00, **MPU9250_GYRO_SELF_TEST_Y** = 0x01,
  MPU9250_GYRO_SELF_TEST_Z = 0x02 }
- enum MPU9250_MAG_REGISTER_t {
  MPU9250_MAG_REG_DEVICE_ID = 0x00, MPU9250_MAG_REG_INFORMATION = 0x01, MPU9250_MAG_REG_STATUS_ = 0x02, MPU9250_MAG_REG_X_DATA_LSB = 0x03,
  MPU9250_MAG_REG_X_DATA_MSB = 0x04, MPU9250_MAG_REG_Y_DATA_LSB = 0x05, MPU9250_MAG_REG_Y_DATA = 0x06, MPU9250_MAG_REG_Z_DATA_LSB = 0x07,
  MPU9250_MAG_REG_Z_DATA_MSB = 0x08, MPU9250_MAG_REG_STATUS_2 = 0x09, MPU9250_MAG_REG_CONTROL = 0x0A, MPU9250_MAG_REG_SELF_TEST = 0x0C,
  MPU9250_MAG_REG_I2C_DISABLE = 0x0F, MPU9250_MAG_REG_X_SENSITIVITY = 0x10, MPU9250_MAG_REG_Y_SEN = 0x11, MPU9250_MAG_REG_Z_SENSITIVITY = 0x12 }
- enum MPU9250_MAG_CONTROL_t {
  MPU9250_MAG_STATUS_1_DATA_READY_BIT = 0x00, MPU9250_REG_INT_PIN_BYPASS = 0x37,
  MPU9250_PASSTHROUGH_ON = 0x02, MPU9250_PASSTHROUGH_OFF = 0x00,
  MAG_MODE_SINGLE_MEAS = 0b0001, MAG_MODE_CONTINUOUS_8HZ = 0b0010, MAG_MODE_CONTINUOUS_100HZ = 0b0011, MAG_MODE_POWER_DOWN = 0b0000,
  MAG_MODE_SELF_TEST = 0b1000, MAG_MODE_BITMASK = 0x0F, MAG_MODE_14_BIT = 0x00,
  MAG_MODE_16_BIT = 0x10,
  MAG_MASK_DATA_OVERRUN = 0x02, MAG_MASK_DATA_READY = 0x01, MAG_MASK_DATA_OVERFLOW = 0x08, MAG_MASK_DATA_BIT_RESOLUTION = 0x10 }

Variables

- enum { ... } **MPU9250_ACCEL_REGISTER_t**

5.1.1 Detailed Description

Register map and configuration data for the MPU9250 IMU on the TSLPB V3.

Author

Nicholas Counts

Date

06/19/18

5.1.2 Enumeration Type Documentation

5.1.2.1 MPU9250_MAG_CONTROL_t

```
enum MPU9250_MAG_CONTROL_t
```

Enumerator

MPU9250_REG_INT_PIN_BYPASS	READ/WRITE: Allow passthrough mode.
MPU9250_PASSTHROUGH_ON	When asserted, the i2c_master interface pins go into 'bypass mode' when the i2c master interface is disabled. The pins will float high due to the internal pull-up if not enabled and the i2c master interface is disabled.
MPU9250_PASSTHROUGH_OFF	& with current register
MAG_MODE_SINGLE_MEAS	Single Measurement Mode.
MAG_MODE_CONTINUOUS_8HZ	Continuous register update mode (8 Hz)
MAG_MODE_CONTINUOUS_100HZ	Continuous register update mode (100 Hz)
MAG_MODE_POWER_DOWN	Low power standby mode.
MAG_MODE_SELF_TEST	Perform a self test with internal magnetic field generator.
MAG_MODE_BITMASK	bit mask for mode-setting register
MAG_MODE_14_BIT	bit 4 off for 14-bit output
MAG_MODE_16_BIT	bit 4 on for 16-bit output
MAG_MASK_DATA_OVERRUN	ST1 bit mask for data overrun.
MAG_MASK_DATA_READY	ST1 bit mask for data ready.
MAG_MASK_DATA_OVERFLOW	ST2 bit mask for "Magnetic sensor overflow occurred" - true if true.
MAG_MASK_DATA_BIT_RESOLUTION	ST2 bit mask: 0 if 14-bit output, 1 if 16-bit output.

5.1.2.2 MPU9250_MAG_REGISTER_t

```
enum MPU9250_MAG_REGISTER_t
```

Enumerator

MPU9250_MAG_REG_DEVICE_ID	READ: Device ID.
MPU9250_MAG_REG_INFORMATION	READ: Information.
MPU9250_MAG_REG_STATUS_1	READ: Data status.
MPU9250_MAG_REG_X_DATA_LSB	READ: X-axis data (LSB)
MPU9250_MAG_REG_X_DATA_MSB	READ: X-axis data (MSB)
MPU9250_MAG_REG_Y_DATA_LSB	READ: Y-axis data (LSB)
MPU9250_MAG_REG_Y_DATA_MSB	READ: Y-axis data (MSB)
MPU9250_MAG_REG_Z_DATA_LSB	READ: Z-axis data (LSB)
MPU9250_MAG_REG_Z_DATA_MSB	READ: Z-axis data (MLSB)
MPU9250_MAG_REG_STATUS_2	READ: Data Status.
MPU9250_MAG_REG_CONTROL	READ/WRITE: Mode Setting.
MPU9250_MAG_REG_SELF_TEST	READ/WRITE:
MPU9250_MAG_REG_I2C_DISABLE	READ/WRITE:
MPU9250_MAG_REG_X_SENSITIVITY	READ:
MPU9250_MAG_REG_Y_SENSITIVITY	READ:
MPU9250_MAG_REG_Z_SENSITIVITY	READ:

5.2 NSL_ThinSat.h File Reference

Function prototypes, includes, and definitions for NSL to TSLPB Arduino interface.

Macros

- #define NSL_PACKET_SIZE 38

  Total bytes in the TSL Payload Packet.
- #define NSL_PACKET_HEADER_LENGTH 3

  Total Bytes.
- #define NSL_PACKET_HEADER {0x50, 0x50, 0x50}

  The 3 byte preabmpe to NSL Payload Packets.
- #define NSL_BAUD_RATE 38400

  From ETSat_Payload_ICD_v5.9.pdf page 10.
- #define **NSL_SERIAL_ACK** {0xAA, 0x05, 0x00}
- #define **NSL_SERIAL_NAK** {0xAA, 0x05, 0xFF}
- #define NSL_SERIAL_READY LOW

  The NSL Mothership is able to receive a payload data packet.
- #define NSL_SERIAL_BUSY HIGH

  The NSL Mothership is unable to receive a payload data packet.

5.2.1 Detailed Description

Function prototypes, includes, and definitions for NSL to TSLPB Arduino interface.

Author

Nicholas Counts

Version

0.6.0

Date

06/12/18

```
This header is used by TSLPB.h and TSLPB.cpp to define the interface
to the NSL Mothership.
```

5.3 ThinSat_DataPacket_generic.h File Reference

Defines the standard data structure used to store the user's payload data, and the union that is used to transmit the data to the NSL Mothership. Users must define ThinSat_DataPacket_custom_h, write their own UserDataStruct_t definition, and include the ThinsatPacket_t union typedef.

```
#include "NSL_Thinsat.h"
```

Classes

- struct UserDataStruct_t

  A generic data structure to hold any data the user intends to send back to Earth.
- union ThinsatPacket_t

  A union of the UserDataStruct_t payloadData and a byte array that is used to send the user's mission data to the NSL Mothership.

5.3.1 Detailed Description

Defines the standard data structure used to store the user's payload data, and the union that is used to transmit the data to the NSL Mothership. Users must define ThinSat_DataPacket_custom_h, write their own UserDataStruct_t definition, and include the ThinsatPacket_t union typedef.

Author

Nicholas Counts

Date

06/20/18

5.4 TSLPB.cpp File Reference

Implementation of TSLPB interface for Arduino.

```
#include "TSLPB.h"
```

5.4.1 Detailed Description

Implementation of TSLPB interface for Arduino.

Author

Nicholas Counts

Date

05/15/19

5.5 TSLPB.h File Reference

Function prototypes, includes, and definitions for TSLPB Arduino interface.

```
#include "WProgram.h"
#include "avr/sleep.h"
#include "Wire.h"
#include <SoftwareSerial.h>
#include "NSL_ThinSat.h"
#include "ThinSat_DataPacket_generic.h"
#include "MPU9250_REGS.h"
```

Classes

- class TSLPB

  The controller class for the TSL Payload Board. Create an instance of this class to use its member functions for accessing the onboard analog and digital sensors. Methods for communicating with the NSL Mothership are also included.

Macros

- #define TSL_DIAGNOSTIC_BAUD 9600

  TSLPB diagnostic serial port baud rate.
- #define TSL_NSL_BUS_STATUS_PIN 4

  NSL Serial Busy Line monitoring pin.
- #define TSL_NSL_BUS_RX_PIN 3

  Pin for software serial connectino to NSL Serial Bus.
- #define TSL_NSL_BUS_TX_PIN 5

  Pin for software serial connectino to NSL Serial Bus.
- #define TSL_ADC A7

  ADC reading the MUX_Output.
- #define TSL_MUX_A 7

  Mux A - TSLPB pin number.
- #define TSL_MUX_B 8

  Mux B - TSLPB pin number.
- #define TSL_MUX_C 9

  Mux C - TSLPB pin number.
- #define TSL_MUX_RESPONSE_TIME 10

  10 miliseconds to change
- #define TSL_SENSOR_READY_TIMEOUT 100

  number of milliseconds to wait for an I2C device to become ready
- #define LMA_TEMP_REG_UNUSED_LSBS 5

  TSLPB Digital Temperature Sensor (LMA75A) Macros.
- #define LMA_TEMP_REG_SIGN_BIT 9

  The bit that contains indicates the sign. 0-based.
- #define LMA_TEMP_REG_DEGREES_PER_LSB 0.125

  Temperature resolution in ℃ per LSb.

Enumerations

- enum TSLPB_AnalogSensor_t {
  Solar = 0b000, IR = 0b001, TempInt = 0b010, TempExt = 0b011,
  Current = 0b100, Voltage = 0b101 }

  TSLPB Analog Sensor Selection Enum.
- enum TSLPB_I2CAddress_t {
  DT1_ADDRESS = 0x4A, DT2_ADDRESS = 0x4C, DT3_ADDRESS = 0x4D, DT4_ADDRESS = 0x48,
  DT5_ADDRESS = 0x49, DT6_ADDRESS = 0x4B, IMU_ADDRESS = 0x69, MAG_ADDRESS = 0x0C,
  MEM_ADDRESS = 0x50 }

  TSLPB Digital Sensor Address enum. Used by TSLPB private methods to communicate with the digital sensors over I2C.

- enum TSLPB_DigitalSensor_t {
  DT1, DT2, DT3, DT4,
  DT5, DT6, Accelerometer_x, Accelerometer_y,
  Accelerometer_z, Gyroscope_x, Gyroscope_y, Gyroscope_z,
  Magnetometer_x, Magnetometer_y, Magnetometer_z, IMU_Internal_Temp }

  TSLPB Digital Sensor selection Enum. Used as arguments for TSLPB::readDigitalSensor() and TSLPB::readDigitalSensorRaw()
- enum LM75A_REG {
  LM75A_TEMPERATURE = 0x0, LM75A_CONFIGURATION = 0x1, LM75A_T_HYST = 0x2, LM75A_T_OS = 0x3,
  LM75A_PRODUCT_ID = 0x7 }

  TSLPB Digital Temperature Sensor (LMA75A) Register Selection Enum.

5.5.1 Detailed Description

Function prototypes, includes, and definitions for TSLPB Arduino interface.

Author

Nicholas Counts

Version

0.6.0

Date

06/12/18

5.5.2 Macro Definition Documentation

5.5.2.1 LMA_TEMP_REG_UNUSED_LSBS

```
#define LMA_TEMP_REG_UNUSED_LSBS 5
```

TSLPB Digital Temperature Sensor (LMA75A) Macros.

The number of bits to be discarded (from LSb)

5.5.3 Enumeration Type Documentation

5.5.3.1 LM75A_REG

```
enum LM75A_REG
```

TSLPB Digital Temperature Sensor (LMA75A) Register Selection Enum.

Enumerator

LM75A_TEMPERATURE	0x00 Read only
LM75A_CONFIGURATION	0x01 Read/Write
LM75A_T_HYST	0x02 Read/Write
LM75A_T_OS	0x03 Read/Write
LM75A_PRODUCT_ID	0x07 Read only

5.5.3.2 TSLPB_AnalogSensor_t

`enum TSLPB_AnalogSensor_t`

TSLPB Analog Sensor Selection Enum.

Enumerator

Solar	0b000 (Solar Sensor)
IR	0b001 (IR)
TempInt	0b010 (Temp Int)
TempExt	0b011 (Temp Ext)
Current	0b100 (Current)
Voltage	0b101 (Vcc)

5.5.3.3 TSLPB_DigitalSensor_t

`enum TSLPB_DigitalSensor_t`

TSLPB Digital Sensor selection Enum. Used as arguments for TSLPB::readDigitalSensor() and TSLPB::readDigitalSensorRaw()

Enumerator

DT1	Select LM75A DT1.
DT2	Select LM75A DT2.
DT3	Select LM75A DT3.
DT4	Select LM75A DT4.
DT5	Select LM75A DT5.
DT6	Select LM75A DT6.
Accelerometer_x	Select MPU-9250 Accelerometer x-axis.
Accelerometer_y	Select MPU-9250 Accelerometer y-axis.
Accelerometer_z	Select MPU-9250 Accelerometer z-axis.
Gyroscope_x	Select MPU-9250 Gyroscope x-axis.
Gyroscope_y	Select MPU-9250 Gyroscope y-axis.
Gyroscope_z	Select MPU-9250 Gyroscope z-axis.
Magnetometer_x	Select MPU-9250 Magnetometer x-axis.
Magnetometer_y	Select MPU-9250 Magnetometer y-axis.
Magnetometer_z	Select MPU-9250 Magnetometer z-axis.
IMU_Internal_Temp	Select MPU-9250 Internal Temperature.

5.5.3.4 TSLPB_I2CAddress_t

```
enum TSLPB_I2CAddress_t
```

TSLPB Digital Sensor Address enum. Used by TSLPB private methods to communicate with the digital sensors over I2C.

Note

May be used by client code to access any of the I2C devices on the TSLPB. (with caution!)

Enumerator

DT1_ADDRESS	LM75A.
DT2_ADDRESS	LM75A.
DT3_ADDRESS	LM75A.
DT4_ADDRESS	LM75A.
DT5_ADDRESS	LM75A.
DT6_ADDRESS	LM75A.
IMU_ADDRESS	MPU-9250.
MAG_ADDRESS	MAGNETOMETER I2C Address (slave on the MPU-9250)
MEM_ADDRESS	EEPROM I2C Address for the Microchip 24LC256.

5.6 VCSFA_ThinSat_DataPacket.h File Reference

Defines the custom data structure used to store the user's payload data, and the union that is used to transmit the data to the NSL Mothership.

```
#include "NSL_ThinSat.h"
```

Classes

- struct UserDataStruct_t

  A generic data structure to hold any data the user intends to send back to Earth.

5.6.1 Detailed Description

Defines the custom data structure used to store the user's payload data, and the union that is used to transmit the data to the NSL Mothership.

Author

Nicholas Counts

Date

06/20/18

Index