# Comms Firmware Integration Documentation

By Sophia Yan

GitHub abs-platform/abs-software

Pull request #32

Table of Contents

[Comms Firmware Integration Documentation 1](#_Toc460002623)

[File Organization Overview 3](#_Toc460002624)

[External Libraries 3](#_Toc460002625)

[abs.ino 4](#_Toc460002626)

[comms.cpp & comms.h 4](#_Toc460002627)

[hdlc.cpp & hdlc.h 4](#_Toc460002628)

[Changes Over Summer 5](#_Toc460002629)

## File Organization Overview

A comms library was developed for the communication module of the ABS project. This summer, the library and testing was integrated into one folder.

The integrated folder includes the following files:

abs.h

abs.ino

abs\_assets.ino

abs\_comms.cpp

abs\_comms.h

abs\_hdlc.cpp

abs\_hdlc.h

The files are related in the following way:

|  |
| --- |
| External Libraries  abs.ino  comms.cpp  &  comms.h  com  hdlc.cpp  &  Hdlc.h  assets.ino |

### External Libraries

The external libraries that need to be imported are TimerOne and USB\_Host\_Shield\_Library\_2.0. The comms code uses the external libraries to interact with the phone and hardware. These libraries need to be imported to run the final firmware.

### abs.ino

This is the main file where the final code will be written. The file also includes precompiler case statements to alternate between different test cases during the development phase.

### comms.cpp & comms.h

The two files incorporate the comms system functionalities. The header file includes necessary files, declares registers, declares parameters, defines local variables, and declares functions. The cpp file implements the functions declared in the header file.

The public functions of comms are the following:

|  |  |
| --- | --- |
| Function | Description |
| void configure(); | Configures the transceiver chip to default parameters. Parameters include but not limited to PLLLOOP,FREQ,TXPRATE, MODULATION, etc. |
| void tx(uint8\_t \*data, int data\_size); | Write to a PWRMODE register and configure the transmit frequency. Power on the transceiver to transmit and power it off at the end. |
| char \*rx(); | Configure transceiver to turn switch to receive mode, receive the signal on the designated frequency and turn off again. |
| void change\_x(int parameter, int value); | /\*It has to be specified to the HWmod\_comms. Parameters defined in comms.h\*/  Return a different value depending on the specified parameter. The parameter is passed to the function as a variable |
| unsigned int read\_register(byte this\_register); | Return the result of the register to read |
| void write\_register(byte this\_register, byte this\_value); | Write a value to a specified register |

The registers used in comm.cpp & comms.h are defined in section 3.2 of the AXEM5042 Programming Manual.

### hdlc.cpp & hdlc.h

The hdlc files support the communication protocol and determines how the data is sent to the transceiver. A flow chart and more information can be found on the AXSEM AX5042 Programming Manual pages 29-32.

## Changes Over Summer

Two libraries need to be imported from the library manager on the Arduino IDE to run Comm\_Test.ino. The two libraries are TimerOne and USB\_Host\_Shield\_Library\_2.0.

After importing the libraries there were some compilation errors where the function used in the c file did not match the function declared in the header file. The functions implemented more parameters than the function declaration in the header. Therefore, the header file is altered to include the extra parameters used in the functions. The functions that were altered are shown in the following two code segment.   
  
Before:

|  |
| --- |
| …  33 uint32\_t configure\_DATARATE(int bitrate);  34 void configure\_TMGGAIN(uint32\_t fskmul, uint32\_t datarate, int tmgcorrfrac);  35 void configure\_AGCATTACK(int bitrate);  36 void configure\_AGCDECAY(int bitrate);  37 void configure\_PHASEGAIN();  38 void configure\_FREQGAIN();  … |

After:

|  |
| --- |
| …  33 uint32\_t configure\_DATARATE(int bitrate, int cicdec, int fskmul);  34 void configure\_TMGGAIN(uint32\_t fskmul, uint32\_t datarate, int tmgcorrfrac);  35 void configure\_AGCATTACK(int bitrate, uint8\_t modValue);  36 void configure\_AGCDECAY(int bitrate, uint8\_t modValue);  37 void configure\_PHASEGAIN(uint8\_t modValue);  38 void configure\_FREQGAIN(uint8\_t modValue);  … |

The first step after confirming both abs.ino and comms\_test.ino compiles, is to integrate the two into one final folder called abs.   
  
The goal of integrating was to develop a way to run ‘unit tests’ in the same folder as the final code. This is done by using precompiler case statements #if, #elif, and #else. Only the main.ino file was changed to accommodate the integration of the code. The supporting header and c files were added to the same folder as abs.ino to run the program.   
  
The structure is as follows. Change the value of variable test\_version to move between the different test cases and the final code. The default case is the final, integrated code.

|  |
| --- |
| /\*define constants and macros\*/  /\* set up stuff and initialize variables\*/  #if test\_version == COMMS\_TEST  #else  #endif  Void setup(){  #if test\_version == COMMS\_TEST  //do all the comms test code setup    #elif test\_version == ABS\_TEST  //do all the abs test code setup    #else  //place final integrated code setup here    #endif  };  void loop(){  #if test\_version == COMMS\_TEST  //do all the comms test code    #elif test\_version == ABS\_TEST  //do all the abs test code    #else  //place final integrated code here    #endif  };  # if test\_version == COMMS\_TEST:  /\*support functions for comms\_test\*/  Function 1  Function 2  #elif test\_version == ABS\_TEST  /\*support functions for abs\_test\*/  Function 1  Function 2  #else  /\*support functions for the final version\*/  Function 1  Function 2  #endif |