Comms Firmware Integration Documentation

By Sophia Yan

GitHub abs-platform/abs-software

Pull request #32

Table of Contents

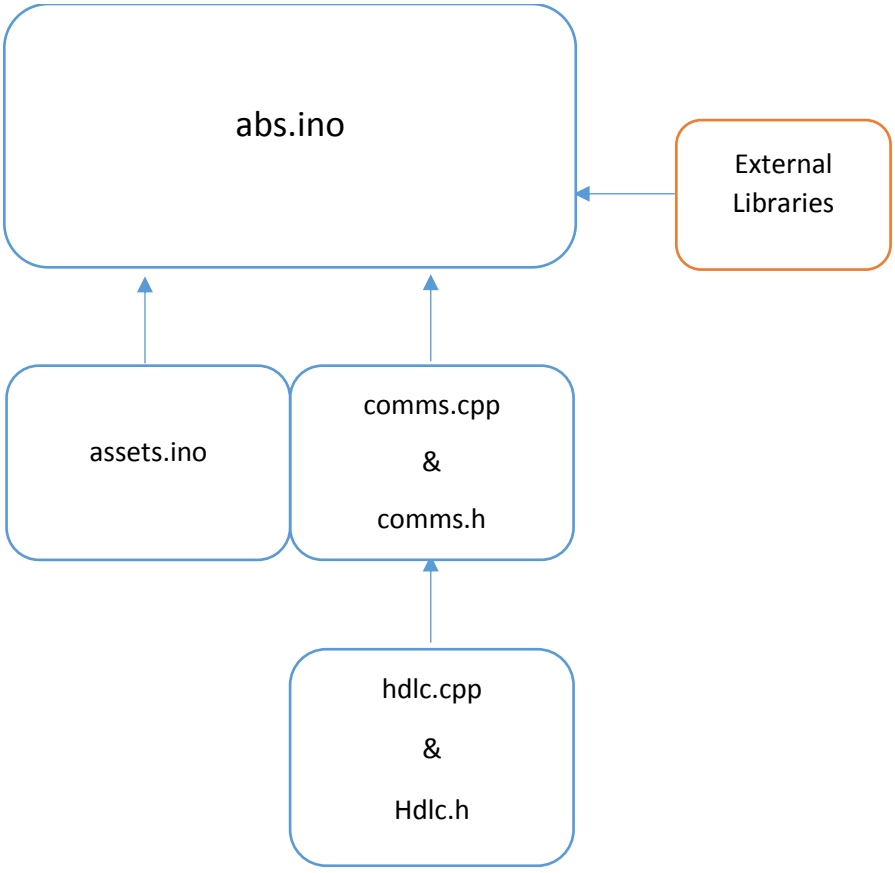
File Organization Overview

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

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
```