Project 4: Library Development

In this project your team will partner with 2 other teams to develop code libraries for peripherals that can be utilized to accomplish future projects.

Each team will develop a library of functions that support one of the following devices:

1. LCD & Button Shield
2. GPS Module
3. IMU Module (BNO055)

Teams should work together to define library interfaces and functionality that support this project and upcoming projects.

The functionality of the libraries should include:

LCD & Button Shield:
- Display formatted text (Words and Numbers)
- Clear the Display
- Read the state of the buttons.

GPS Module:
- Retrieve current GPS Coordinates & Satellite Fix Accuracy
- Calculate the distance between 2 GPS Points
- Calculate the north-oriented angle between 2 GPS Points

IMU Module:
- Calibrate the IMU Module
- Write Calibration to IMU
- Read the current Euler Vectors.

The demo for this project will consist of 2 parts:
1. Demonstration of individual library functionality.
2. Demonstration of integrated application using all 3 libraries.

The specific **demonstration of individual library functionality** is up to you, but the functionality described above should be demonstrated in some way.

The **integrated application demonstration** will follow this procedure.

1. The IMU should be calibrated prior to demonstration.

2. On reset, the LCD should display ("WAITING FOR FIX") until the GPS unit has an accurate satellite fix.
3. Once a fix is acquired, the LCD should display "READY" on the first line, "SELECT DEST" on the second line.
4. The hardware will be carried to a location, and the Select button will be pressed. The program should save the current GPS location as the "Destination". The LCD display should show "DEST SAVED" for 1 second.
5. The hardware will be carried to various other locations. The distance in meters, current heading, and desired heading to the destination location should be displayed on the LCD.

Your team should submit 2 projects - your library and test project, and your integrated application project.

Submission

Once you have completed the development challenge, export your project file by clicking File -> Export, then choose Archive File. Click "Browse" to save your project as "CSE325_Project_4_*yourlastname*.zip"

Submit the following to canvas:

- **Zipped project file**
- Completed **Integrity.txt**

Demonstration

You must submit your code and complete a demonstration with a TA by the due date set on the course website. The TA will read your code, and you must demonstrate that it works. The TA may ask you questions about your code, and you must be prepared to answer.

Grading

Rubric:

Criteria	**Points**
Library is well structured, well commented, variables and methods are named clearly.	20
Library functionality is complete and appropriate for current and future applications.	20

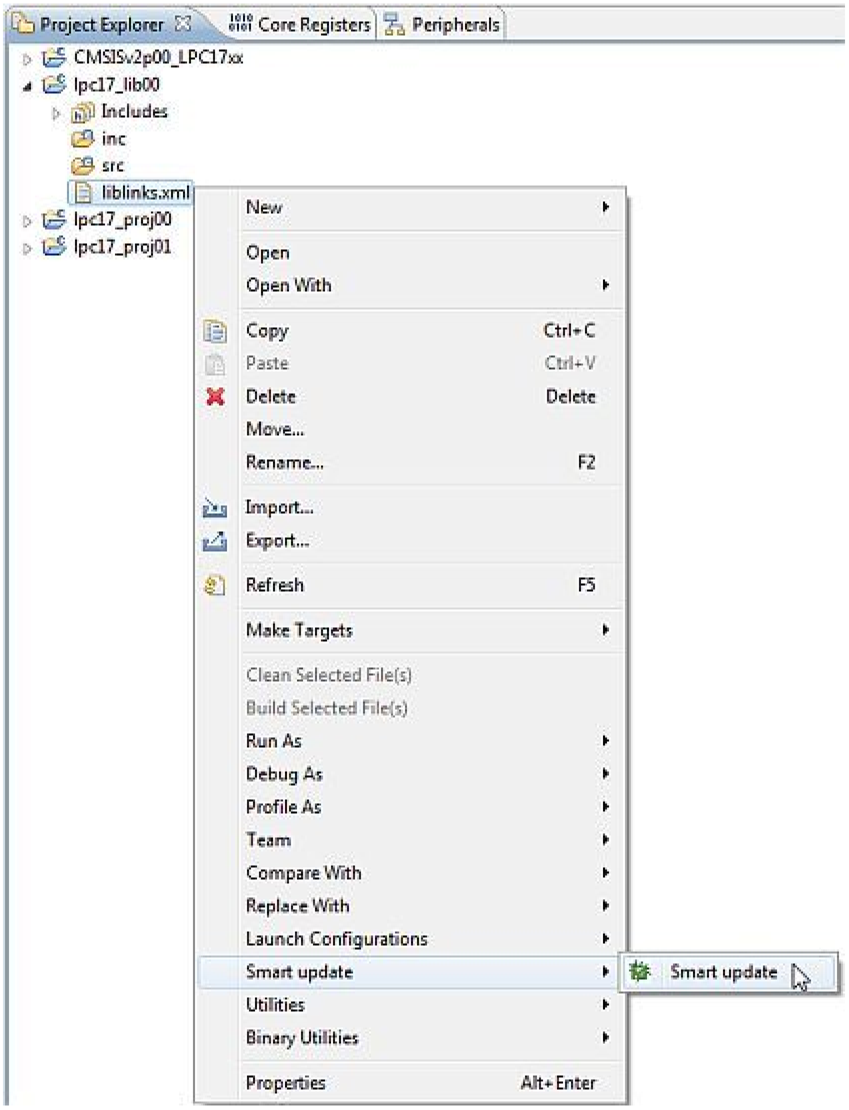
Library Demonstration shows required functionality	20
Integrated application demonstration shows required functionality	20
The student can answer questions about the program correctly.	20
Total	**100**

If you do not upload your code before the assignment deadline, you will receive a grade of 0.

Making a Library Project

In the Quickstart Menu, select "New Project", and then the KL46Z Freedom board. In the configure project view, select "C Static Library", then give the project a name, and click finish to create the project.

To use the library in a different project, right click the "liblinks.xml" file in the library project, and select Smart Update.



Choose the projects that you would like the library to be included. You'll now be able to #include the header files from the library project in the application projects, and the library will be compiled into the application executables.

DEVELOPMENT HINTS

LCD Library

- Check the LCD Schematic for Pin assignments.
- See Page 46 of the LCD Display Datasheet for initialization procedure.
- Implement Clear, SetCursor commands.
- Implement Print String, Print Integer, and Print Float functions.

GPS Library

- Use UART1 on PTE0 and PTE1, (UART0 is used for SDA)
- UART1 clock source is the Bus Clock (Not selectable)
- Configure for 9600 Baud, 8-N-1
- Configure module for NMEA Output, RMC & GGA sentences at 1Hz
- Use an interrupt for receiving characters.
- Use double-buffering for receiving and parsing messages.
- Include string.h and use strchr to tokenize and parse messages into appropriate fields.
- Note that raw strings report latitude and longitude as Degrees Minutes Seconds (DDDMMSS.SSSSS) and decimal seconds, as a concatenated numerical string.
- Process latitude and longitude into a fixed point representation (such as decimal Lat & Lon multiplied by 10,000, with North Positive, South Negative, West Negative, East Positive

IMU Library

- Use the I2C1 interface on PTC1 and PTC2
- Device Initialization Process:
    - Send Reset Command, Wait 1000ms
    - Set Mode: CONFIG, Wait 100ms
    - Reset Page
    - Set Normal Power Mode, Wait 100ms
    - Set Mode: NDOF, Wait 100ms
    - Set External Clock Source, Wait 100ms
    - Read Chip ID, should be 0xA0.
- Calibration Process:
    - After initialization, move the platform according to datasheet instructions. Read the Calibration Status Register repeatedly until SYS, GYR, ACC, & MAG fields all read 3. Maintain this for a few seconds.
    - Print the calibration offset registers to the console on a button press
    - To read the calibration offset values:
        - Switch to CONFIG Mode, wait 100ms
        - Read the calibration offset value
        - Switch back to NDOF mode, wait 100ms
- Re-using calibration offsets
    - Copy offset values into code.
    - Switch mode to CONFIG
    - Write offset registers
    - Switch mode to NDOF mode, wait 100ms.
    - Read System Status register, move platform around until SYS reads "3".
- Correct usage of the KL46Z I2C interface is under-documented. Use the function outlines below to develop your communication functions:

initI2C()

```
        // Enable Clock Gating for I2C module and Port
        // Setup Pin Mode for I2C
        // Write 0 to all I2C registers
        // Write 0x50 to FLT register (clears all bits)
        // Clear status flags
        // Set I2C Divider Register (Choose a clock frequency less than 100 KHz
        // Set bit to enable I2C Module

clearStatusFlags()
        // Clear STOPF and Undocumented STARTF bit 4 in Filter Register
        // Clear ARBL and IICIF bits in Status Register
TCFWait()
        // Wait for TCF bit to Set in Status Register

IICIFWait()
        // Wait for IICIF bit to Set in Status Register

SendStart()
        // Set MST and TX bits in Control 1 Register

RepeatStart()
        // Set MST, TX and RSTA bits in Control 1 Register
        // Wait 6 cycles

SendStop()
        // Clear MST, TX and TXAK bits in Control 1 Register
        // Wait for BUSY bit to go low in Status Register

clearIICIF()
        // Clear IICIF bit in Status Register

RxAK()
        // Return 1 if byte has been ACK'd. (See RXAK in Status Register)


I2C WriteByte (Register Address, Data) {
        clearStatusFlags();
        TCFWait();
        SendStart();

        // TODO: Write Device Address, R/W = 0 to Data Register

        IICIFWait();
        if (!RxAK()){
                printf("NO RESPONSE - Address");
                SendStop();
                return;
        }

        clearIICIF();

        // TODO: Write Register address to Data Register
        IICIFWait();
        if (!RxAK()){
                printf("NO RESPONSE - Register");
                SendStop();
                return;
        }

        TCFWait();
        clearIICIF();

        // Write Data byte to Data Register

        IICIFWait();
        if (!RxAK()){
                printf("Incorrect ACK - Data");
        }

        clearIICIF();
        SendStop();
}

Read Block(Register Address, Destination Data Byte Array, Length) {
        unsigned char dummy = 0;
        clearStatusFlags();
        TCFWait();
        SendStart();

        dummy++;  // Do something to suppress the warning.

        //TODO: Write Device Address, R/W = 0 to Data Register

        IICIFWait();
```

```
if (!RxAK()){
        printf("NO RESPONSE - Address");
        SendStop();
        return 0;
}

clearIICIF();

// Write Register address to Data Register
IICIFWait();
if (!RxAK()){
        printf("NO RESPONSE - Register");
        SendStop();
        return 0;
}

clearIICIF();
RepeatStart();

// Write device address again, R/W = 1 to Data Register
IICIFWait();
if (!RxAK()){
        printf("NO RESPONSE - Restart");
        SendStop();
        return 0;
}

TCFWait();
clearIICIF();

// Switch to RX by clearing TX and TXAK bits in Control 1 register

if(len==1){
        // Set TXAK to NACK in Control 1 - No more data!
}

dummy = I2C1->D; // Dummy Read

for(int index=0; index<len; index++){
        IICIFWait();
        clearIICIF();

        if(/*Second to Last Byte*/){
                // Set TXAK to NACK in Control 1 - No more data!
            }

            if(/*Last Byte*/) == 1){
                SendStop();
            }
            // Read Byte from Data Register into Array
        }
    }
```