Nathan Bunnell
Embedded Controller Programming with C
ECE-40291
Student ID: U08895857

Date: 5/29/2021

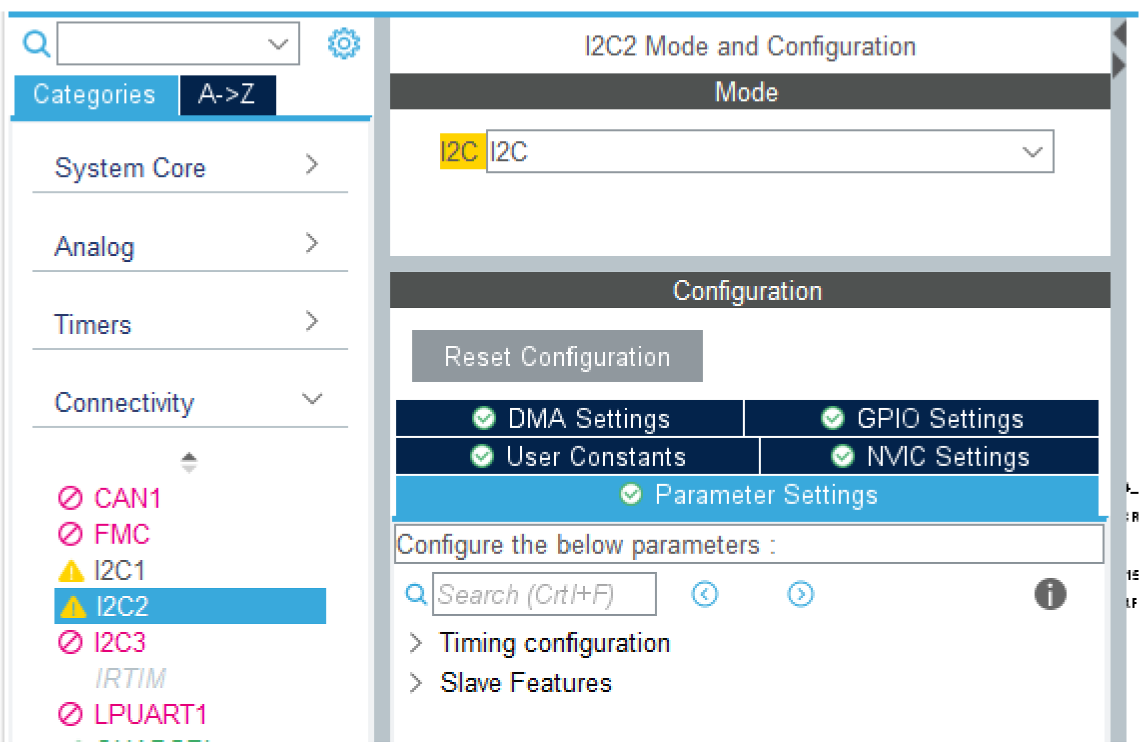
Assignment 7: I2C

The following will document completion of the eighth assignment for ECE-40291, with the stated goals of:

1. Your skills for using HAL I2C APIs to read the I2C sensor of your choice on the STM32 Discovery Board. There are several I2C sensors on the STM32 Discovery Board --- pick one that is of interest to you, and then read data using the I2C HAL APIs.
2. Display your data from your sensor on the UART1 console.

1. Your skills for using HAL I2C APIs to read the I2C sensor of your choice on the STM32 Discovery Board. There are several I2C sensors on the STM32 Discovery Board --- pick one that is of interest to you, and then read data using the I2C HAL APIs.

As in all previous assignments, open and generate the default project configuration for the Disco board. To leverage the on board I2C2 bus, we'll also need to open the device configuration tool, and enable it under 'Connectivity' >> 'I2C2', as seen here. The default options are satisfactory for this project's scope.



Nathan Bunnell
Embedded Controller Programming with C
ECE-40291
Student ID: U08895857

Date: 5/29/2021

From here, the IDE will handle the addition of the appropriate init calls and other support files; we can focus on the target sensors and associated firmware from here on. For this project I chose to expand on the HTS221 sensor used in the previous assignments for two reasons. First: to attempt readings with the onboard humidity sensor in addition to reading temperature, and second: to dive further into the hardware, specifically the calibration routines. This also had the benefit of allowing some code re-use (with much expansion) from the ADC assignment in ECE-40293, which made the rest of this assignment fairly simple to implement after enough time spent in datasheets and tech notes. Additionally, I chose to read the ambient pressure from the LPS22HB module as a supplement to the taking “the easy way out” with code reused from another course. Finally, I had hoped to incorporate an RTC module on the external I2C bus broken out over the Arduino connectors and while I was able to perform basic read/write operations using my I2C Driver Mini[1] debugging tool over my PC, I decided not to incorporate it due to time constraints with regards to the course end date approaching. On the interface level, I again reused existing code to deploy a simple CLI over UART 1, with a list of options for each sensor broken out, followed by status info and the process values as they were read back from each data point.

So, jumping into *main.c*, we’ll first take care of the boiler-plate items: including the standard libraries needed for string and print functions, followed by defining the bus addresses of each module, taken from the Disco board’s published schematics.

```c
/* Private includes ----------------------------
/* USER CODE BEGIN Includes */

#include <stdio.h>
#include <string.h>

/* USER CODE END Includes */
```

```c
// Define R/W addresses for temp sensor
#define HTS221_READ_ADDRESS 0xbf
#define HTS221_WRITE_ADDRESS 0xbe
#define LPS22HB_READ_ADDRESS 0xbb
#define LPS22HB_WRITE_ADDRESS 0xba

/* USER CODE END PD */
```

Coming back to the support functions that do all of the processing later, we jump into the *main()* function, insert calls to *HTS221_pwr_en()* and *HTS221_get_cal_data()*, then print out the CLI header and jump into the *while(1)* loop, where the CLI options are printed and the user can input their selection. As it has been covered multiple times elsewhere, I won’t go into greater detail beyond that on the CLI structure other than to segue into the support function definitions that each option calls. Full project code will be submitted with this report.

Option one, "HTS221 read request" will call function *HTS221_get_sensor_data()*. First though, lets jump back to the earlier setup calls to *HTS221_pwr_en()* and *HTS221_get_cal_data()*. *HTS221_pwr_en()* is an extremely simple "wake up" style function that enable the module into one-shot mode and allows us to take readings later.

```
static void HTS221_pwr_en(void)
{
    // Configure control register 1 (CTRL_REG1, 0x20) bit 7 to enable one-shot
    uint8_t ctrlReg1 = 0x20;
    uint8_t CTRL_REG2_Value[] = {ctrlReg1, (1 << 7)};

    // Send the target register to the device
    HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, CTRL_REG2_Value, sizeof(CTRL_REG2_Value), 1000);

}

```

HTS221_get_cal_data() is a bit more complicated, and involves reading the calibration values for the temperature and humidity sensors from the module into global variables for use when readings are taken at a later point. This is an example of the deeper dive I wanted to take into this piece of hardware, and while the information made available in the device datasheet was a bit vague, I found an example implementation for the Raspberry Pi written in Python[2], followed by an ST published tech note[3] that fleshed out the process in great detail. Breaking it out below, we pull the appropriate calibration values in from the module and store them for use in the calibration equation defined in the tech note for each data point, occurring later in the *HTS221_get_sensor_data()* call.

6. Compute the RH [%] value, by linear interpolation, applying the formula below:

$$H_{RH}[\%] = \frac{(H1\_rH - H0\_rH) \cdot (H\_T\_OUT - H0\_T0\_OUT)}{H1\_T0\_OUT - H0\_T0\_OUT} + H0\_rH$$

7. Compute the T [degC] value, by linear interpolation, applying the following formula:

$$T[degC] = \frac{(T1\_degC - T0\_degC) \cdot (T\_OUT - T0\_OUT)}{T1\_OUT - T0\_OUT} + T0\_degC$$

Nathan Bunnell
Embedded Controller Programming with C
ECE-40291
Student ID: U08895857

Date: 5/29/2021

```c
static void HTS221_get_cal_data(void)
{
    /***************************************************************************************************************/
    // Request humidity and temperature calibration data stored in registers 0x30 to 0x3F
    // Reference pg. 26 of data sheet (https://www.st.com/resource/en/datasheet/hts221.pdf)
    //   for register names and definitions
    // Reference tech note TN1218 on calibration procedures
    // https://www.st.com/resource/en/technical_note/dm00208001-interpreting-humidity-and-temperature-readings-in-the-hts221-digi

    /***************************************************************************************************************/
    // Humidity calibration values

    // Register H0_rh_x2, address 0x30. Divide register value by 2 for calibration value
    uint8_t H0_rH_Address = 0x30;
    HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, &H0_rH_Address, sizeof(H0_rH_Address), 1000);
    H0_rH_Value = 0xff; // Junk default value
    HAL_I2C_Master_Receive(&hi2c2, HTS221_READ_ADDRESS, (uint8_t *)&H0_rH_Value, sizeof(H0_rH_Value), 1000);
    H0_rH_Value = H0_rH_Value / 2;

    // Register H1_rh_x2, address 0x31. Divide register value by 2 for calibration value
    uint8_t H1_rH_Address = 0x31;
    HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, &H1_rH_Address, sizeof(H1_rH_Address), 1000);
    H1_rH_Value = 0xff; // Junk default value
    HAL_I2C_Master_Receive(&hi2c2, HTS221_READ_ADDRESS, (uint8_t *)&H1_rH_Value, sizeof(H1_rH_Value), 1000);
    H1_rH_Value = H1_rH_Value / 2;

    // Register H0_T0_OUT, addresses 0x36 and 0x37
    uint8_t H0_T0_OUT_Address = 0x36 | 0x80;
    HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, &H0_T0_OUT_Address, sizeof(H0_T0_OUT_Address), 1000);
    H0_T0_OUT_Value = 0xffff; // Junk default value
    HAL_I2C_Master_Receive(&hi2c2, HTS221_READ_ADDRESS, (uint8_t *)&H0_T0_OUT_Value, sizeof(H0_T0_OUT_Value), 1000);

    // Register H1_T0_OUT, addresses 0x3A and 0x3B
    uint8_t H1_T0_OUT_Address = 0x3A | 0x80;
    HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, &H1_T0_OUT_Address, sizeof(H1_T0_OUT_Address), 1000);
    H1_T0_OUT_Value = 0xffff; // Junk default value
    HAL_I2C_Master_Receive(&hi2c2, HTS221_READ_ADDRESS, (uint8_t *)&H1_T0_OUT_Value, sizeof(H1_T0_OUT_Value), 1000);

    /***************************************************************************************************************/
    // Temperature calibration values

    // Register T0_degC_x8, address 0x32. Divide register value by 8 for calibration value
    uint8_t T0_degC_Address = 0x32;
    HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, &T0_degC_Address, sizeof(T0_degC_Address), 1000);
    T0_degC_Value = 0xff; // Junk default value
    HAL_I2C_Master_Receive(&hi2c2, HTS221_READ_ADDRESS, (uint8_t *)&T0_degC_Value, sizeof(T0_degC_Value), 1000);
    T0_degC_Value = T0_degC_Value / 8;

    // Register T1_degC_x8, address 0x33. Divide register value by 8 for calibration value
    uint8_t T1_degC_Address = 0x33;
    HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, &T1_degC_Address, sizeof(T1_degC_Address), 1000);
    T1_degC_Value = 0xff; // Junk default value
    HAL_I2C_Master_Receive(&hi2c2, HTS221_READ_ADDRESS, (uint8_t *)&T1_degC_Value, sizeof(T1_degC_Value), 1000);
    T1_degC_Value = T1_degC_Value / 8;

    // Register T1/T0 msb, address 0x35. Mask bits (0 & 1), (2 & 3) to get values of T0_degC & T1_degC
    uint8_t T1_T0_msb_Address = 0x35;
    HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, &T1_T0_msb_Address, sizeof(T1_T0_msb_Address), 1000);
    uint8_t T1_T0_msb_Value = 0xff; // Junk default value
    HAL_I2C_Master_Receive(&hi2c2, HTS221_READ_ADDRESS, (uint8_t *)&T1_T0_msb_Value, sizeof(T1_T0_msb_Value), 1000);
    T0_degC = (T1_T0_msb_Value && (0b0011));
    T1_degC = (T1_T0_msb_Value && (0b1100));

    // Register T0_OUT, addresses 0x3C and 0x3D
    uint8_t T0_OUT_Address = 0x3C | 0x80;
    HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, &T0_OUT_Address, sizeof(T0_OUT_Address), 1000);
    T0_OUT_Value = 0xffff; // Junk default value
    HAL_I2C_Master_Receive(&hi2c2, HTS221_READ_ADDRESS, (uint8_t *)&T0_OUT_Value, sizeof(T0_OUT_Value), 1000);

    // Register T1_OUT, addresses 0x3C and 0x3D
    uint8_t T1_OUT_Address = 0x3C | 0x80;
    HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, &T1_OUT_Address, sizeof(T1_OUT_Address), 1000);
    T1_OUT_Value = 0xffff; // Junk default value
    HAL_I2C_Master_Receive(&hi2c2, HTS221_READ_ADDRESS, (uint8_t *)&T1_OUT_Value, sizeof(T1_OUT_Value), 1000);

}
```

Nathan Bunnell
Embedded Controller Programming with C
ECE-40291
Student ID: U08895857

Date: 5/29/2021

Jumping next to *HTS221_get_sensor_data()*, we set the module to one-shot mode, then loiter until the STATUS register updates to indicate that the new sample is ready.

```c
static void HTS221_get_sensor_data(void)
{
    // Large char buffer for strings sent over the console
    char buffer[100] = {0};

    // Configure control register 2 (CTRL_REG2, 0x21) bit 0 to enable one-shot
    uint8_t CTRL_REG2_Address = 0x21;
    uint8_t CTRL_REG2_Value[] = {CTRL_REG2_Address, (1 << 0)};

    // Send the target register to the device
    HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, CTRL_REG2_Value, sizeof(CTRL_REG2_Value), 1000);

    // Define status register (STATUS_REG2, 0x27) bit 0 to monitor for new sample available
    uint8_t STATUS_Address = 0x27;
    uint8_t STATUS_Value = 0;

    // Print status message to console
    snprintf(buffer, sizeof(buffer), "\tRequesting new sample...");
    HAL_UART_Transmit(&huart1, (uint8_t*) buffer, strlen(buffer), 1000);

    // Loiter for a bit to allow time for conversion to complete and be made available
    uint8_t count = 0;
    while (count < 10)  // arbitrary "long enough" delay value
    {
        // Send the address of the status register
        HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, &STATUS_Address, sizeof(STATUS_Address), 1000);

        // Read back the value of the status register
        HAL_I2C_Master_Receive(&hi2c2, HTS221_READ_ADDRESS, (uint8_t *)&STATUS_Value, sizeof(STATUS_Value), 1000);

        // If the new sample is ready, break out of while-loop...
        if (STATUS_Value & 0x01)
        {
            break;
        }

        // Update status message on console with '.' to indicate processing
        snprintf(buffer, sizeof(buffer), ".");
        HAL_UART_Transmit(&huart1, (uint8_t*) buffer, strlen(buffer), 1000);

        // Else wait for a bit, increment the counter, and keep looping
        HAL_Delay(100);
        count++;
    }

```

Nathan Bunnell
Embedded Controller Programming with C
ECE-40291
Student ID: U08895857

Date: 5/29/2021

Moving on to the LPS22HB sensor code, we wrap everything into the *LPS22HB_get_sensor_data()* function call, which is structurally very similar to the HTS version, minus the cal section as that is handled internal to the module. We again configure the device for one-shot mode, wait on the status register to report that it is ready, and then send a request to get the values of 3 registers that represent the 24 bit pressure value. Each one of these is shifted into a single variable with the appropriate bit offset and then divided by 4096 as per the datasheet's example. Finally, this value is printed over the console. I should note that this device also includes a temperature sensor but I elected not to read its data as we already had accomplished that with the HTS and I felt the time would be better spent focusing on other assignments.

Nathan Bunnell
Embedded Controller Programming with C
ECE-40291
Student ID: U08895857

Date: 5/29/2021

Once we see that flag set, we request the values of the humidity and temperature registers then pass both through their respective calibration and scaling formula before printing the final results to the console.

```c

    // Read the values of the humidity register H_OUT, address 0x28 and 0x29
    uint8_t H_OUT_Address = 0x28 | 0x80;
    HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, &H_OUT_Address, sizeof(H_OUT_Address), 1000);
    int16_t H_OUT_Value = 0xbeef; // Junk default value
    HAL_I2C_Master_Receive(&hi2c2, HTS221_READ_ADDRESS, (uint8_t *)&H_OUT_Value, sizeof(H_OUT_Value), 1000);

    // Calculate and print value of humidity in %rH.
    int16_t humidityValue = (((H1_rH_Value - H0_rH_Value) * (H_OUT_Value - H0_T0_OUT_Value))/(H1_T0_OUT_Value - H0_T0_OUT_Value)) + (H0_rH_Value);
    snprintf(buffer, sizeof(buffer), "\n\tHumidity: %d%%rH\n", humidityValue);
    HAL_UART_Transmit(&huart1, (uint8_t*) buffer, strlen(buffer), 1000);


    // Read the values of the temperature register T_OUT, address 0x2A and 0x2B
    uint8_t T_OUT_Address = 0x28 | 0x80;
    HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE_ADDRESS, &T_OUT_Address, sizeof(T_OUT_Address), 1000);
    int16_t T_OUT_Value = 0xbeef; // Junk default value
    HAL_I2C_Master_Receive(&hi2c2, HTS221_READ_ADDRESS, (uint8_t *)&T_OUT_Value, sizeof(T_OUT_Value), 1000);

    // Calculate and print value of temperature in degC.
    int16_t temperatureValue = (((T1_degC_Value - T0_degC_Value) * (T_OUT_Value - T0_OUT_Value))/(T1_OUT_Value - T0_OUT_Value)) + (T0_degC_Value);
    snprintf(buffer, sizeof(buffer), "\tTemperature: %ddegC\n", temperatureValue);
    HAL_UART_Transmit(&huart1, (uint8_t*) buffer, strlen(buffer), 1000);

}

```

2. Display your data from your sensor on the UART1 console.

With all code developed, compiled and flashed without errors or warnings, we can open PuTTY and are presented with the CLI. Selecting option 1 results in the below:

COM3 - PuTTY

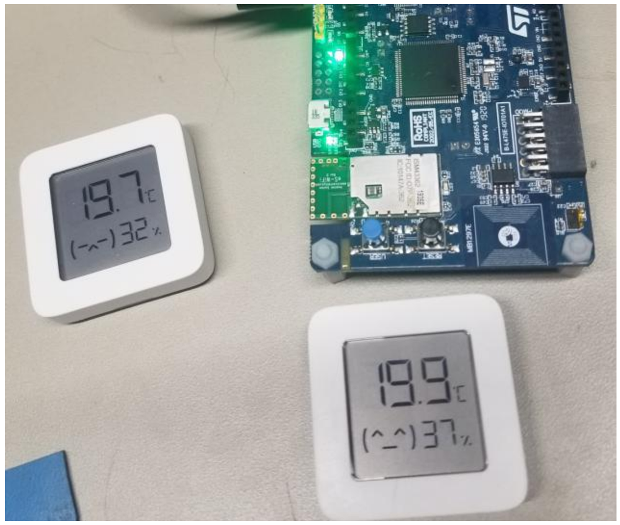
```
simpleCLI Interface v0.4
------------------------------
Options:
        1: HTS221 read
        2: LPS22HB read
        3: RTC read (not implemented)

$> 1

HTS221 read request:
        Requesting new sample...
        Humidity: 28%rH
        Temperature: 20degC
```

As in the other course, I sanity checked this reading against my handy temp/ RH monitors and found that this value was within a reasonable range to their, especially considering that

Nathan Bunnell
Embedded Controller Programming with C
ECE-40291
Student ID: U08895857

Date: 5/29/2021

they're not the most accurate sensors to begin with; for example: note the difference in RH even between the two units.



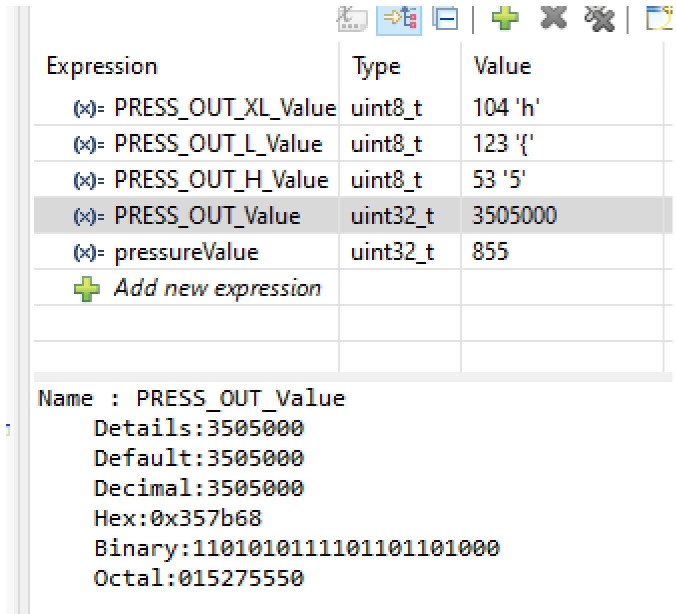
Selecting option 2, we get:

```
Options:
        1: HTS221 read
        2: LPS22HB read
        3: RTC read (not implemented)

$> 2

LPS22HB read request
        Requesting new sample....
        Pressure: 855hPa
```

I'll say up front: this number seems **very** low compared to what I would have expected. It is entirely possible (likely?) I missed a step in the process outlined in the datasheet but not having another pressure reference handy to compare with I reviewed my code process and then compared the values of each register to those in the final converted variable in the debugger and confirmed that they at least were correct. So in the example below, the value of *PRESS_OUT_Value* is 0x357b68, and the values of *PRESS_OUT_XL, PRESS_OUT_L,* & *PRESS_OUT_H* were 0x68, 0x7b, & 0x35, respectively. So I can say with confidence that my data handling is correct, even if I might be missing something in the overall process. Alternatively, my basement could just be at an extremely low pressure.

Nathan Bunnell
Embedded Controller Programming with C
ECE-40291
Student ID: U08895857

Date: 5/29/2021



Closing Thoughts

As in previous assignments in this course, having completed something similar in the sibling course made this task overall fairly easy. I greatly enjoyed jumping into the datasheets for both modules and working out the specific steps needed to pull data out. I really wish I'd had time to incorporate the RTC module but am already considering how that would be a nice feature to implement in the final coffee make project and I think I can see the code developed here finding a way to be reused in that task.