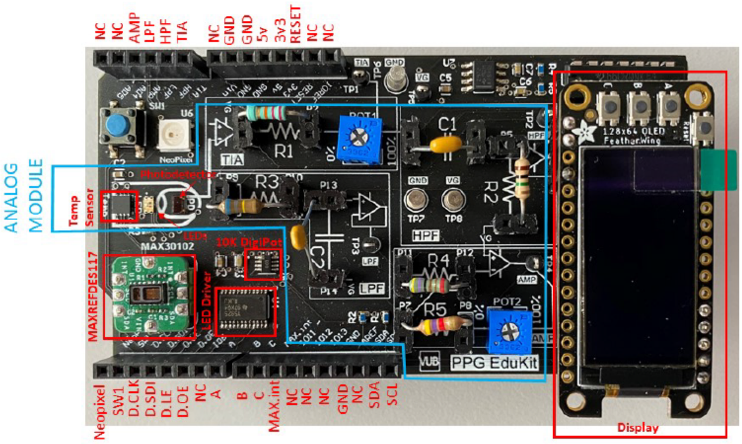
Lab 1: Interface I2C devices in PPG EduKit platform with PSOC.

Introduction

This lab helps the user to get a first interaction with the PPG EduKit platform configuring the display driver and the body temperature sensor. The goal is to be able to integrate the OLED display library and to write a new driver for the temperature sensor.

The PPG EduKit platform is shown below. The module can be used with the CY8CPROTO-063-BLE board using the bridge adaptor provided. The OLED display has integrated the SH1107 driver [1]. SH1107 is a single-chip CMOS OLED/PLED driver that includes a controller for organic/polymer light emitting diode dot-matrix graphic display system. SH1107 consists of 128 segments, 128 commons that can support a maximum display resolution of 128 x 128. The temperature sensor (MAX30205) is a chip from Maxim which provides accurate human body temperature readings with an accuracy of 0.1 °C. Both devices can be interfaced over I2C.



Objectives

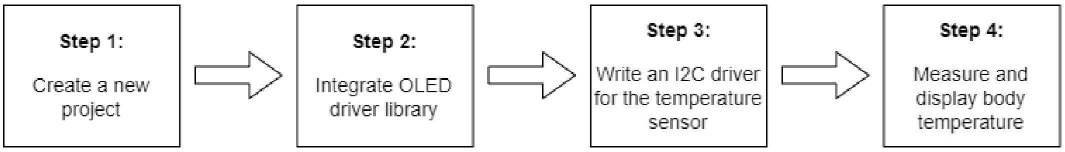
After completing this lab, you will be able to:

- Understand how to set up a PSOC Creator project
- Integrate the OLED display library
- Write an I2C driver for the MAX30205 sensor

Procedure

This lab is separated into steps that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

The lab includes 4 primary steps: create a project using PSOC Creator for CY8CPROTO-063-BLE board, integrate the OLED display software library, write a driver for the MAX30205 sensor, combine both libraries and display the body temperature on the OLED display.



Step 1: Create a new project

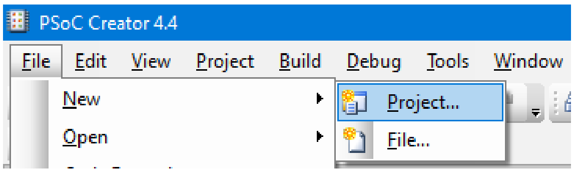
Step 2: Integrate OLED driver library

Step 3: Write an I2C driver for the temperature sensor

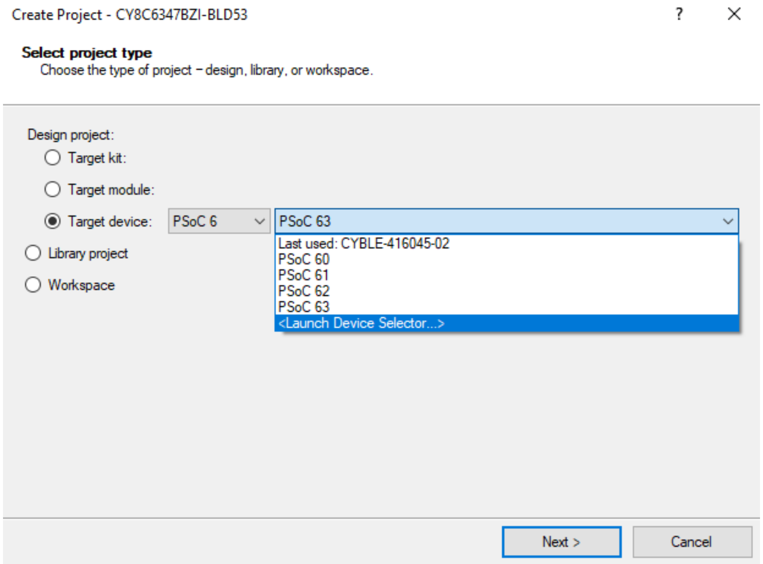
Step 4: Measure and display body temperature

1 Creation of the Project

- Open PSoC Creator
- Go to File → New → Project



- Select the target device → PSOC6 → <Launch Device Selector>



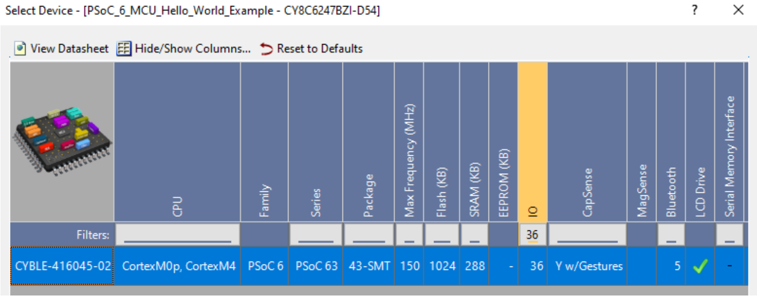
Create Project - CY8C6347BZI-BLD53

Select project type
Choose the type of project – design, library, or workspace.

Design project:
○ Target kit:
○ Target module:
● Target device: PSoC 6 | PSoC 63
○ Library project
○ Workspace

Last used: CYBLE-416045-02
PSoC 60
PSoC 61
PSoC 62
PSoC 63
<Launch Device Selector...>

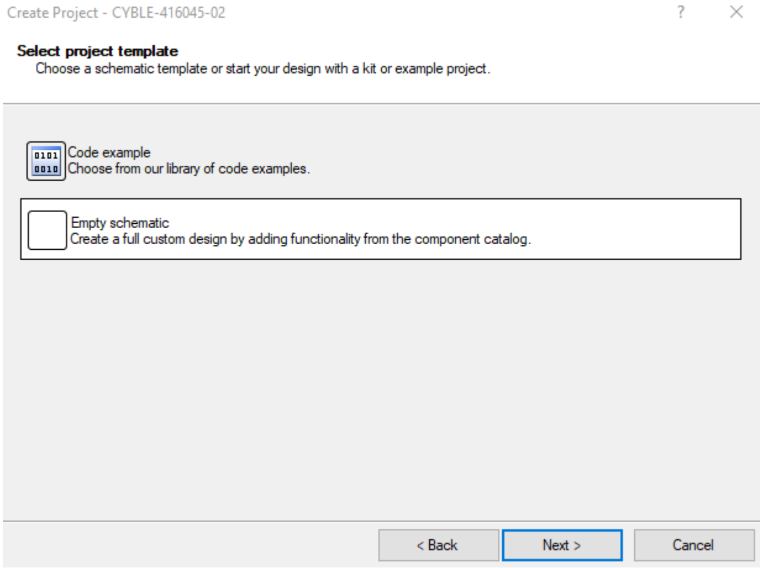
- Select the **CYBLE-416045-2** Device



Select Device - [PSoC_6_MCU_Hello_World_Example - CY8C6247BZI-D54]

	CPU	Family	Series	Package	Max Frequency (MHz)	Flash (KB)	SRAM (KB)	EEPROM (KB)	IO	CapSense	MagSense	Bluetooth	LCD Drive	Serial Memory Interface
Filters:									36					
CYBLE-416045-02	CortexM0p, CortexM4	PSoC 6	PSoC 63	43-SMT	150	1024	288	-	36	Y w/Gestures		5	✓	-

- Select "Empty schematic" and click "Next"



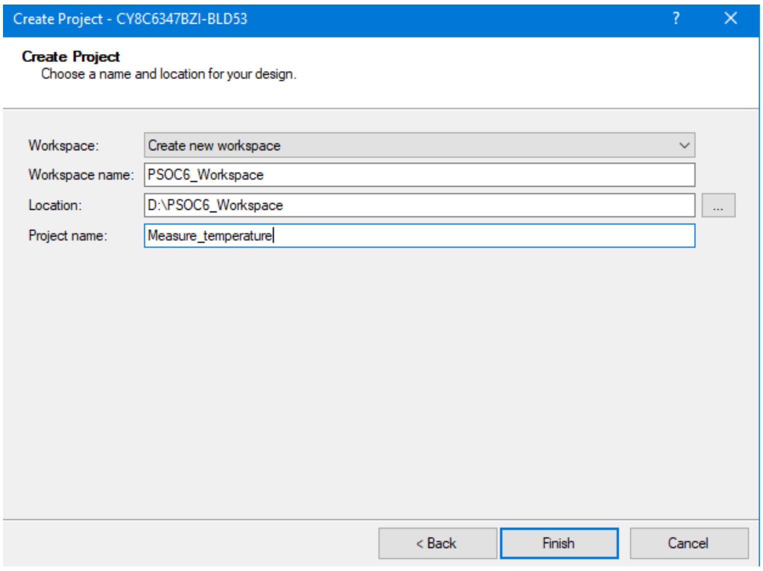
Create Project - CYBLE-416045-02

Select project template
Choose a schematic template or start your design with a kit or example project.

Code example
Choose from our library of code examples.

Empty schematic
Create a full custom design by adding functionality from the component catalog.

- In the "Select target IDE(s)" window, click "Next"
- Create a new Workspace and the project name is **Measure_temperature**



Create Project - CY8C6347BZI-BLD53

Create Project
Choose a name and location for your design.

Workspace: Create new workspace
Workspace name: PSOC6_Workspace
Location: D:\PSOC6_Workspace
Project name: Measure_temperature

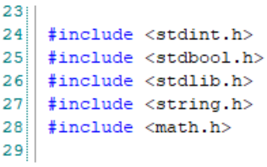
2 Integrate OLED driver library

The next step is to integrate the software driver that controls the OLED display. Adafruit provides a C++ library for monochrome OLEDs based on SH110X drivers and a graphical library for LCD and OLED displays. The OLED library provided with the PPG EduKit is a modified version of the original C++ library and is provided as a C library.

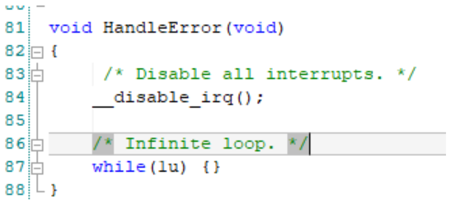
Before importing the OLED library, the user has to import the utils library provided in the lab.

Utils library

The library imports all the common libraries for all the PPG EduKit libraries and defines a common error handler.

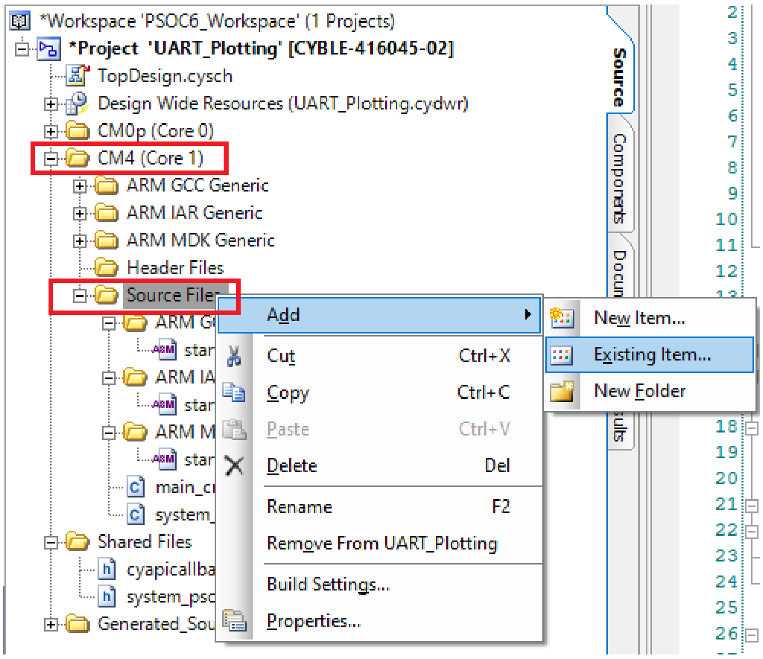


```c

#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

```



```c
void HandleError(void)
{
    /* Disable all interrupts. */
    __disable_irq();

    /* Infinite loop. */
    while(1u) {}
}
```

- Import the source file (utils.c) in CM4 (Core1) → Source Files

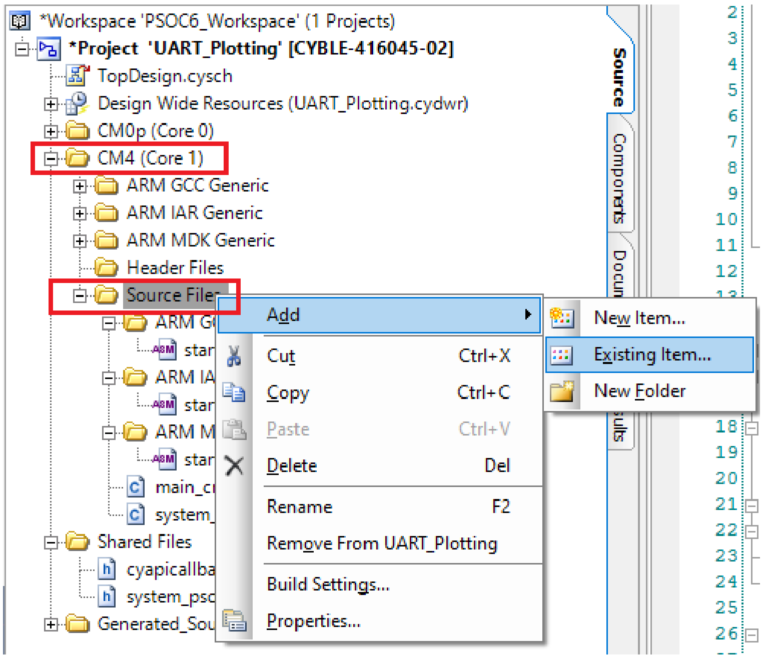


- Repeat the process for the header file (utils.h). Import the file in CM4 (Core1) → Header Files

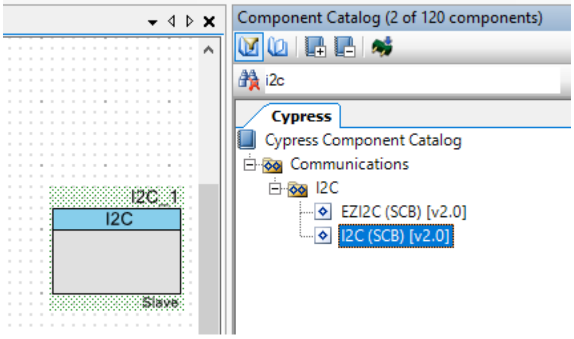
OLED library

The library merges the GFX library (graphical) and the SH110X library into one file. The graphical library provides a common syntax and set of graphics functions for all of our LCD and OLED displays and LED matrices. The SH110X library is a driver library for monochrome displays that have integrated the SH1107 or SH1106G drivers.

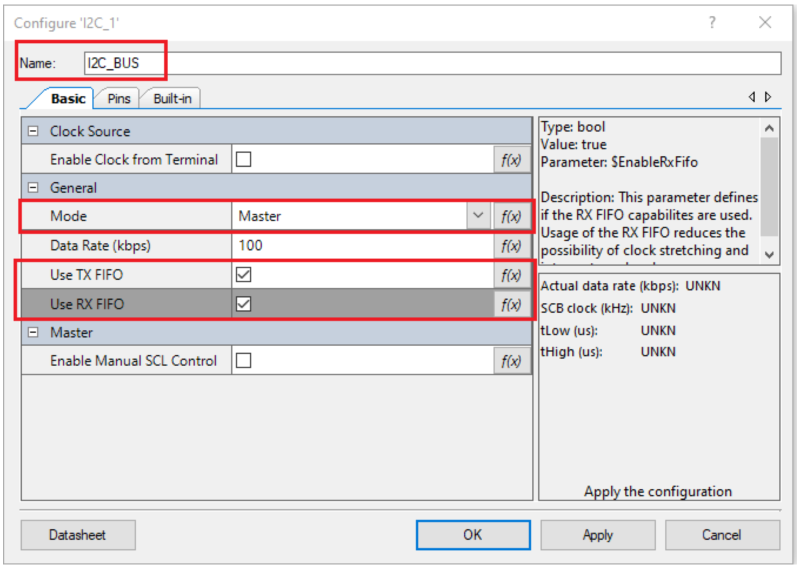
- Import the source file (**oled_driver.c**) in CM4 (Core1) → Source Files



- Repeat the process for the header file (**oled_driver.h**) and for **font.h** file. Import the files in CM4 (Core1) → Header Files
- Go to the **TopDesign** schematic. In the component catalog (right panel at the right of the screen), write **I2C** and drag and drop the component into the schematic.



- Configure the I2C component by double-click on it. Rename the component as **I2C_BUS**, set the mode as **master** and enable **TX/RX FIFO** buffers.



Name: I2C_BUS

⊟ Clock Source	
Enable Clock from Terminal	□
⊟ General	
Mode	Master
Data Rate (kbps)	100
Use TX FIFO	☑
Use RX FIFO	☑
⊟ Master	
Enable Manual SCL Control	□

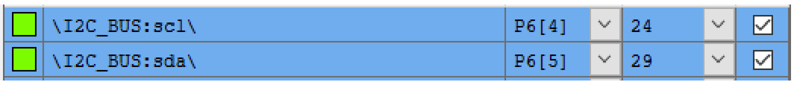
Type: bool
Value: true
Parameter: $EnableRxFifo

Description: This parameter defines if the RX FIFO capabilites are used. Usage of the RX FIFO reduces the possibility of clock stretching and

Actual data rate (kbps): UNKN
SCB clock (kHz): UNKN
tLow (us): UNKN
tHigh (us): UNKN

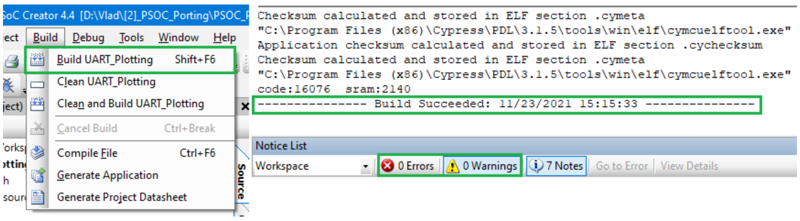
Apply the configuration

- Go to Design Wide Resources → Pins and assign the I2C SCL and SDA pins as follows:



🟩	\I2C_BUS:scl\	P6[4]	24	☑
🟩	\I2C_BUS:sda\	P6[5]	29	☑

- Build the project to check if there is any error.



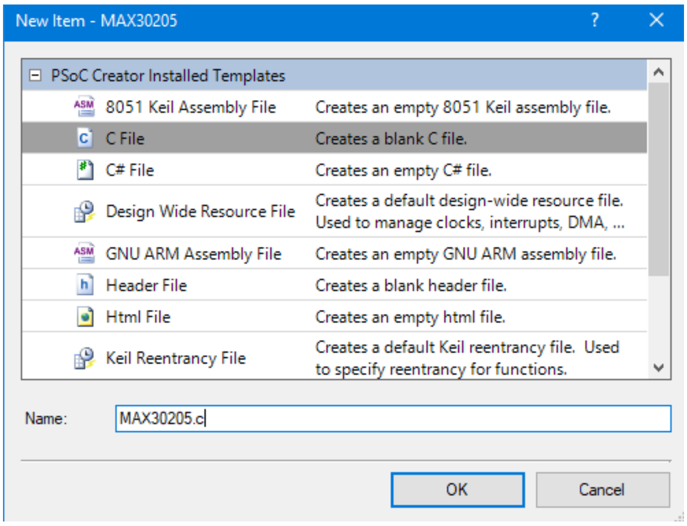
```
Checksum calculated and stored in ELF section .cymeta
"C:\Program Files (x86)\Cypress\PDL\3.1.5\tools\win\elf\cymcuelftool.exe"
Application checksum calculated and stored in ELF section .cychecksum
Checksum calculated and stored in ELF section .cymeta
"C:\Program Files (x86)\Cypress\PDL\3.1.5\tools\win\elf\cymcuelftool.exe"
code:16076  sram:2140
--------------- Build Succeeded: 11/23/2021 15:15:33 ---------------
```

- Take a look at the application programming interface (**oled_library.c**) and try to understand driver functions.

3 Interface MAX30205 temperature sensor

The body temperature sensor can be interfaced through an I2C-compatible, 2-wire serial interface. The I2C serial interface accepts standard write byte, read byte, send byte, and receive byte commands to read the temperature data and configure the behavior of the opendrain overtemperature shutdown output. Therefore, the driver should include a write function and a read function that will be at the core of any other functions that have a higher level of abstraction.

- The driver should include 2 files: a source file and a header file. Go to CM4 (Core1) → Source Files and add a new blank C file.



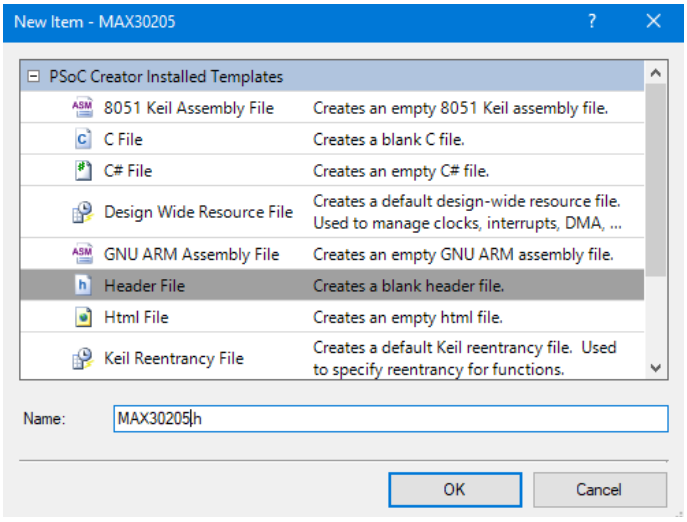
New Item - MAX30205

PSoC Creator Installed Templates

8051 Keil Assembly File	Creates an empty 8051 Keil assembly file.
C File	Creates a blank C file.
C# File	Creates an empty C# file.
Design Wide Resource File	Creates a default design-wide resource file. Used to manage clocks, interrupts, DMA, ...
GNU ARM Assembly File	Creates an empty GNU ARM assembly file.
Header File	Creates a blank header file.
Html File	Creates an empty html file.
Keil Reentrancy File	Creates a default Keil reentrancy file. Used to specify reentrancy for functions.

Name: MAX30205.c

- Go to CM4 (Core1) → Header Files and add a new blank header file.



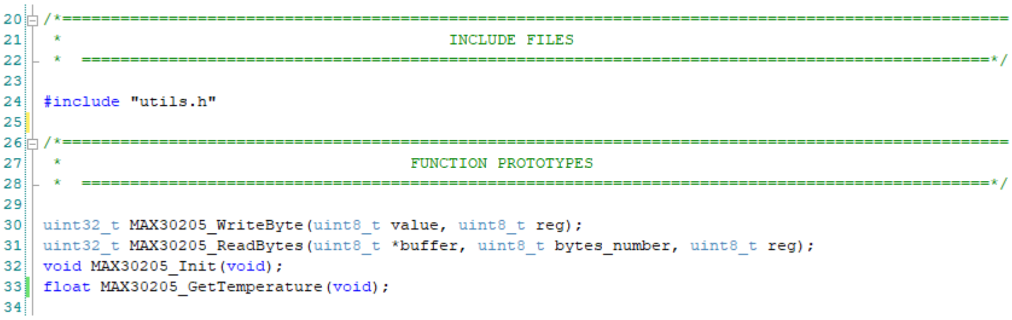
New Item - MAX30205

PSoC Creator Installed Templates

8051 Keil Assembly File	Creates an empty 8051 Keil assembly file.
C File	Creates a blank C file.
C# File	Creates an empty C# file.
Design Wide Resource File	Creates a default design-wide resource file. Used to manage clocks, interrupts, DMA, ...
GNU ARM Assembly File	Creates an empty GNU ARM assembly file.
Header File	Creates a blank header file.
Html File	Creates an empty html file.
Keil Reentrancy File	Creates a default Keil reentrancy file. Used to specify reentrancy for functions.

Name: MAX30205.h

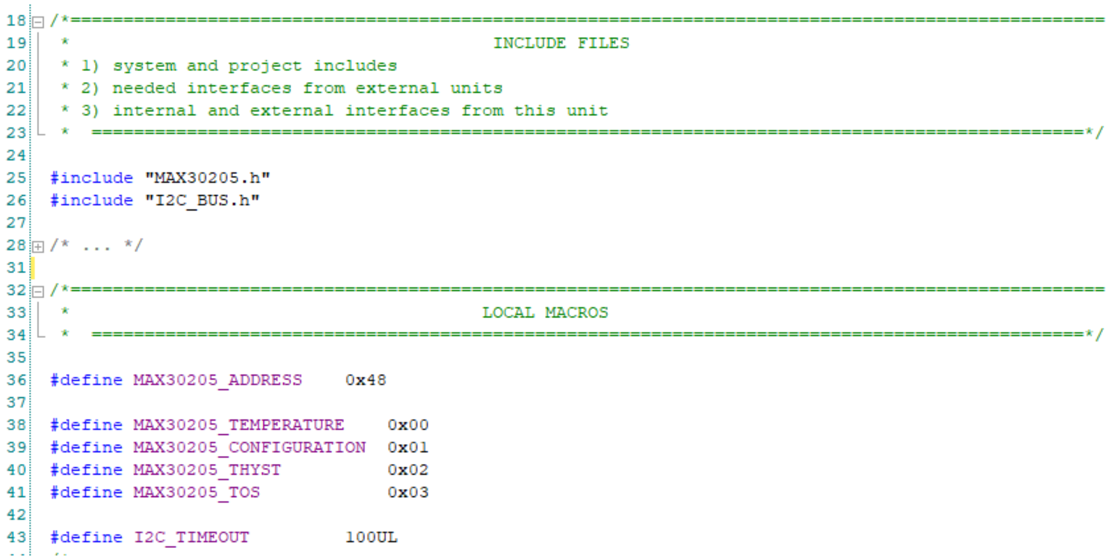
- One should start by defining the driver specifications. The driver should be able to write a byte to the sensor registers, to read multiple bytes at once (temperature is stored as 2 data bytes). Also, the initialization of the device should be done in one function and the temperature reading in another function. Declare the function prototypes inside the header file of the driver as follows:



```
/*==========================================================================================
 *                                      INCLUDE FILES
 * ========================================================================================*/

#include "utils.h"

/*==========================================================================================
 *                                   FUNCTION PROTOTYPES
 * ========================================================================================*/

uint32_t MAX30205_WriteByte(uint8_t value, uint8_t reg);
uint32_t MAX30205_ReadBytes(uint8_t *buffer, uint8_t bytes_number, uint8_t reg);
void MAX30205_Init(void);
float MAX30205_GetTemperature(void);
```

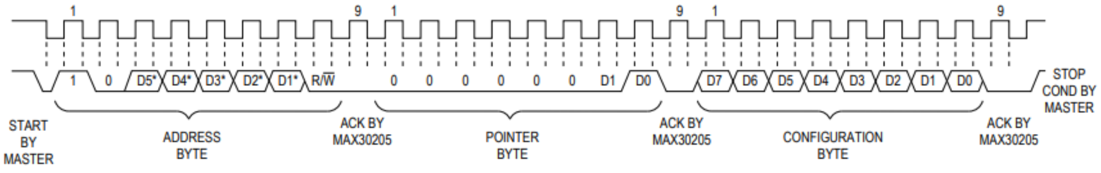
- The I2C address of the device is **48H**. The sensor features three address select lines with a total of 32 available addresses. All the three address select lines are connected to ground (see schematic). According to datasheet [2], the 8-bit address of the device is 90H (10010000b), therefore the 7-bit address is 48H (01001000b) if the 8-bit address is shifter right by 1.

- Go to **MAX30205.c** file. Include the MAX30205 header file and the I2C library header. Define device address, register addresses (Table 2 in the datasheet) and the timeout value.



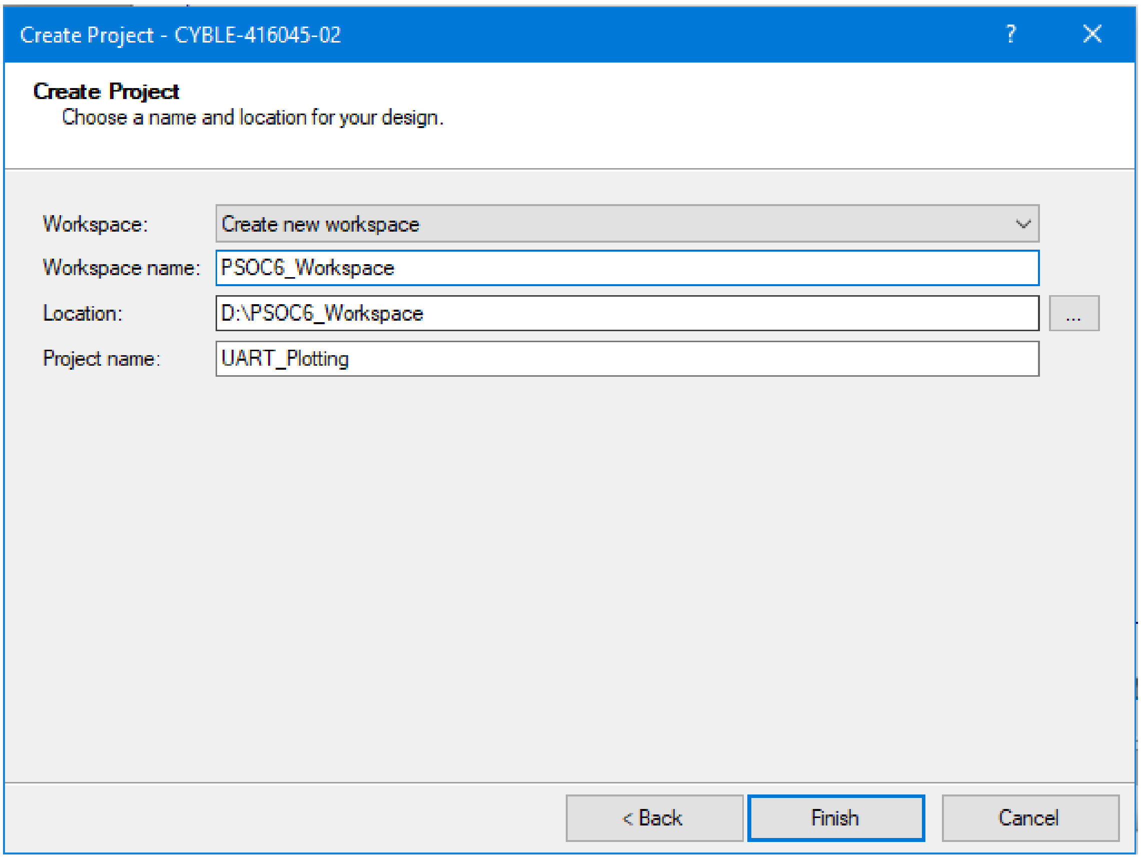
```
/*================================================================================================
 *                                          INCLUDE FILES
 * 1) system and project includes
 * 2) needed interfaces from external units
 * 3) internal and external interfaces from this unit
 *  ==============================================================================================*/

#include "MAX30205.h"
#include "I2C_BUS.h"

/* ... */

/*================================================================================================
 *                                          LOCAL MACROS
 *  ==============================================================================================*/

#define MAX30205_ADDRESS    0x48

#define MAX30205_TEMPERATURE    0x00
#define MAX30205_CONFIGURATION  0x01
#define MAX30205_THYST          0x02
#define MAX30205_TOS            0x03

#define I2C_TIMEOUT         100UL
```

- Before implementing the write function, the I2C timing diagram has to be checked. This information can be found in the datasheet.



(a) CONFIGURATION REGISTER WRITE.

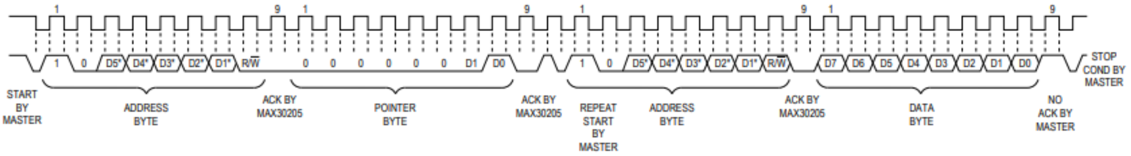
- The write function should follow the timing diagram above. The master should send a start condition over the bus, wait for slave acknowledge, write the register address followed by the value to be written. Then a stop condition on the bus is sent.



Create Project
Choose a name and location for your design.

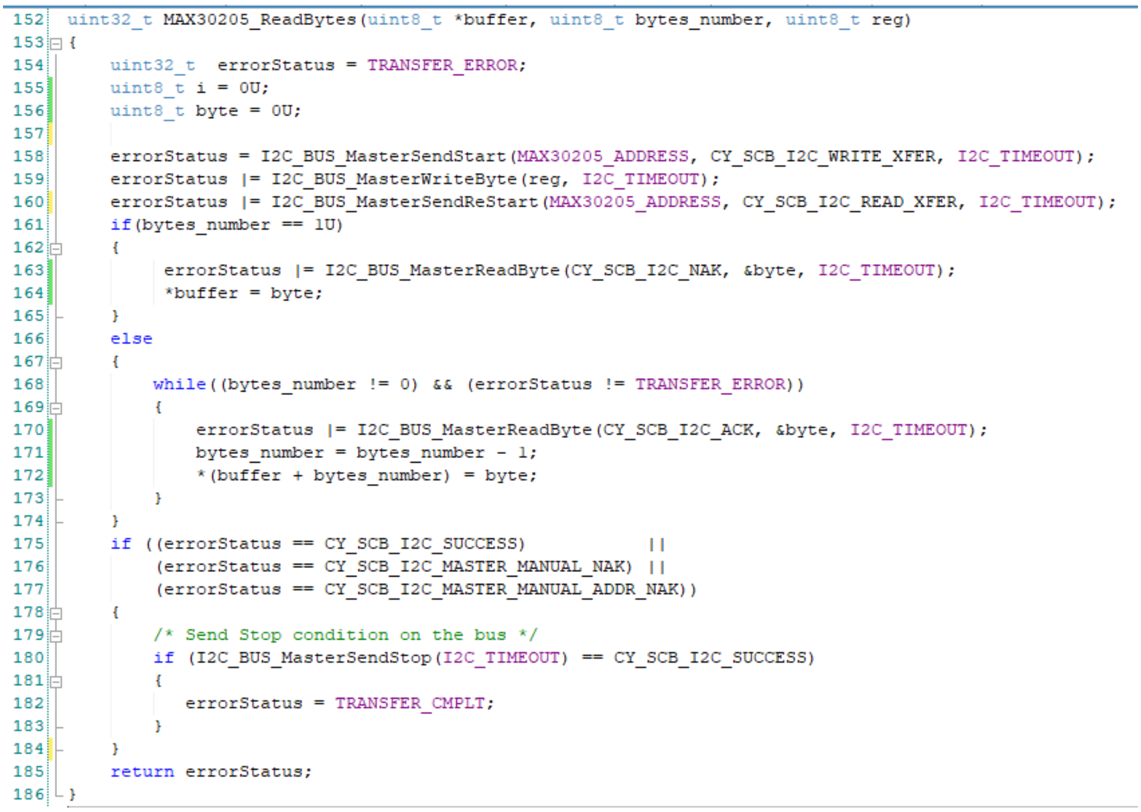
Workspace: Create new workspace
Workspace name: PSOC6_Workspace
Location: D:\PSOC6_Workspace
Project name: UART_Plotting

- Getting the value of the temperature register requires reading two bytes over the bus. Therefore, the function should be able to read a number of bytes higher than 1. To read one byte, the master has to send an not acknowledge signal after the returned data byte.



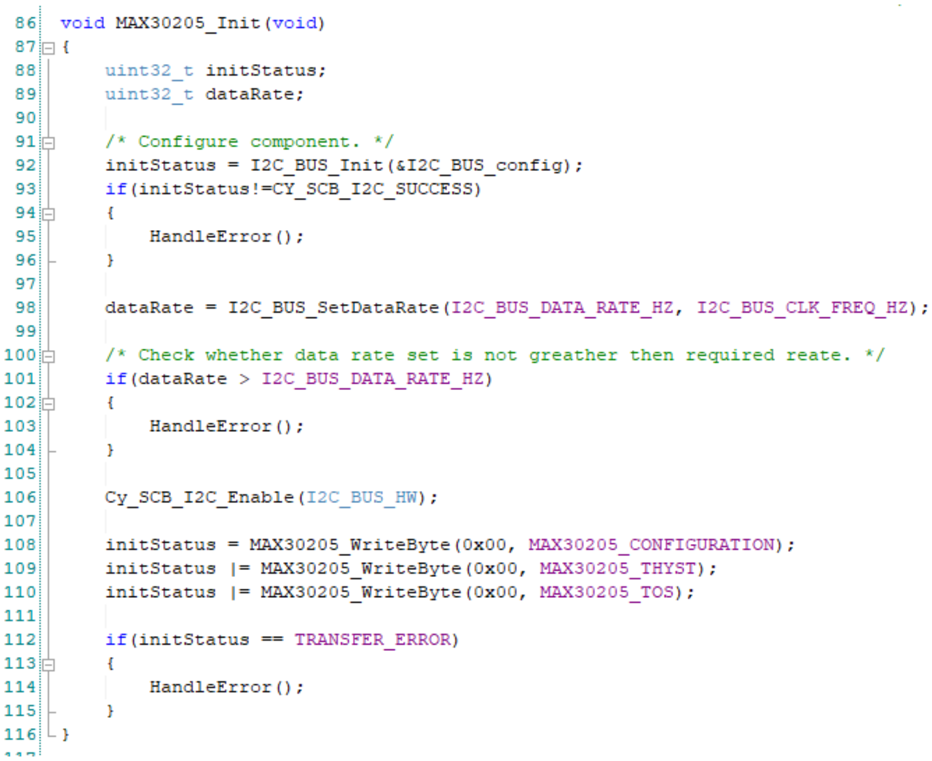
(a) TYPICAL POINTER SET FOLLOWED BY IMMEDIATE READ FROM CONFIGURATION REGISTER.

- The master has to send a start condition over the bus, to write the register address that has to be read, and to send a restart condition. To read multiple bytes the master has to send an acknowledge signal after each returned data byte.



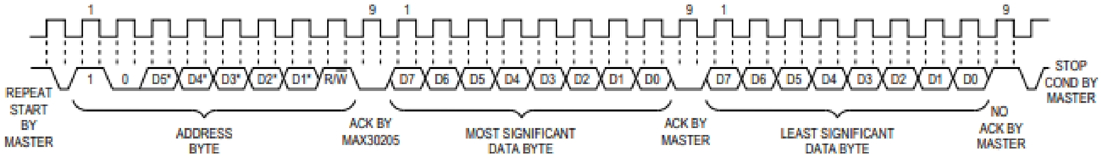
```c
uint32_t MAX30205_ReadBytes(uint8_t *buffer, uint8_t bytes_number, uint8_t reg)
{
    uint32_t  errorStatus = TRANSFER_ERROR;
    uint8_t i = 0U;
    uint8_t byte = 0U;

    errorStatus = I2C_BUS_MasterSendStart(MAX30205_ADDRESS, CY_SCB_I2C_WRITE_XFER, I2C_TIMEOUT);
    errorStatus |= I2C_BUS_MasterWriteByte(reg, I2C_TIMEOUT);
    errorStatus |= I2C_BUS_MasterSendReStart(MAX30205_ADDRESS, CY_SCB_I2C_READ_XFER, I2C_TIMEOUT);
    if(bytes_number == 1U)
    {
        errorStatus |= I2C_BUS_MasterReadByte(CY_SCB_I2C_NAK, &byte, I2C_TIMEOUT);
        *buffer = byte;
    }
    else
    {
        while((bytes_number != 0) && (errorStatus != TRANSFER_ERROR))
        {
            errorStatus |= I2C_BUS_MasterReadByte(CY_SCB_I2C_ACK, &byte, I2C_TIMEOUT);
            bytes_number = bytes_number - 1;
            *(buffer + bytes_number) = byte;
        }
    }
    if ((errorStatus == CY_SCB_I2C_SUCCESS)           ||
        (errorStatus == CY_SCB_I2C_MASTER_MANUAL_NAK) ||
        (errorStatus == CY_SCB_I2C_MASTER_MANUAL_ADDR_NAK))
    {
        /* Send Stop condition on the bus */
        if (I2C_BUS_MasterSendStop(I2C_TIMEOUT) == CY_SCB_I2C_SUCCESS)
        {
           errorStatus = TRANSFER_CMPLT;
        }
    }
    return errorStatus;
}
```

- Before calling any of the functions above, the I2C peripheral has to be initialized. The initialization function initialize the I2C component and to set the sensor registers to 0.

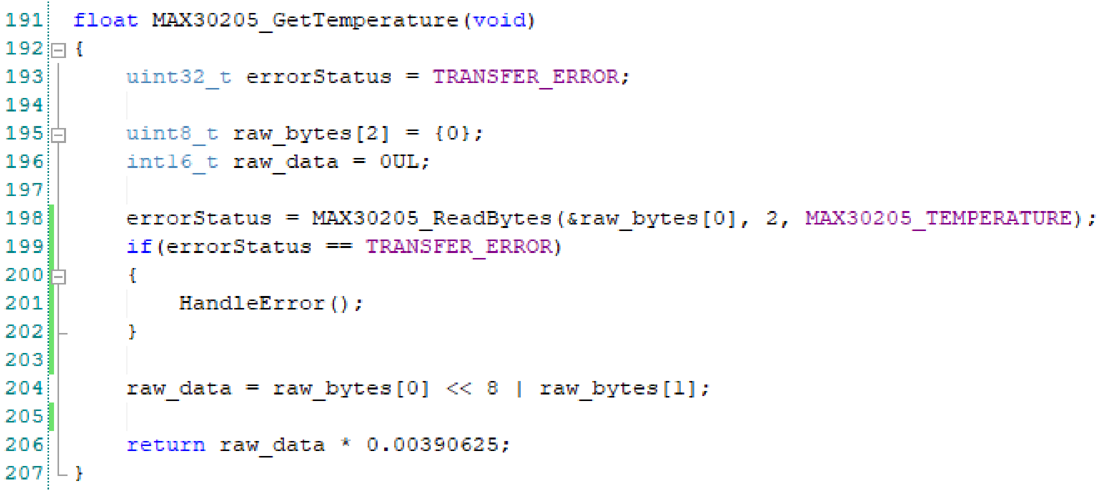


```c
void MAX30205_Init(void)
{
    uint32_t initStatus;
    uint32_t dataRate;

    /* Configure component. */
    initStatus = I2C_BUS_Init(&I2C_BUS_config);
    if(initStatus!=CY_SCB_I2C_SUCCESS)
    {
        HandleError();
    }

    dataRate = I2C_BUS_SetDataRate(I2C_BUS_DATA_RATE_HZ, I2C_BUS_CLK_FREQ_HZ);

    /* Check whether data rate set is not greather then required reate. */
    if(dataRate > I2C_BUS_DATA_RATE_HZ)
    {
        HandleError();
    }

    Cy_SCB_I2C_Enable(I2C_BUS_HW);

    initStatus = MAX30205_WriteByte(0x00, MAX30205_CONFIGURATION);
    initStatus |= MAX30205_WriteByte(0x00, MAX30205_THYST);
    initStatus |= MAX30205_WriteByte(0x00, MAX30205_TOS);

    if(initStatus == TRANSFER_ERROR)
    {
        HandleError();
    }
}
```

- To read the temperature, a read request should be performed for the register 0H (temperature register). According to the datasheet, the value is represented on two bytes, two's complement, and the most significant data byte is sent first. Bits D[15:0] contains the temperature data, with the LSB representing 0.00390625°C and the MSB representing the sign bit.



- The data bytes should be shifted accordingly and the raw temperature value to be multiplied by 0.00390625 (bit resolution).

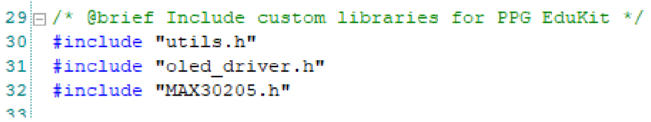


```
float MAX30205_GetTemperature(void)
{
    uint32_t errorStatus = TRANSFER_ERROR;

    uint8_t raw_bytes[2] = {0};
    int16_t raw_data = 0UL;

    errorStatus = MAX30205_ReadBytes(&raw_bytes[0], 2, MAX30205_TEMPERATURE);
    if(errorStatus == TRANSFER_ERROR)
    {
        HandleError();
    }

    raw_data = raw_bytes[0] << 8 | raw_bytes[1];

    return raw_data * 0.00390625;
}
```

4 Create the main program

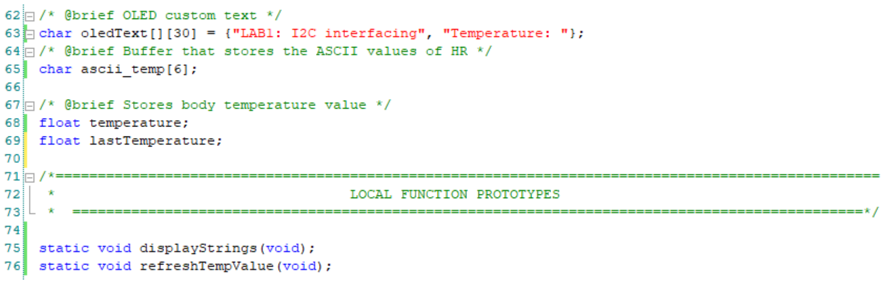
The main program should merge together all the libraries in such a way to reach the goal of displaying the finger temperature on the OLED display.

- Go to **main_c4.c** and include the header file for each software driver.



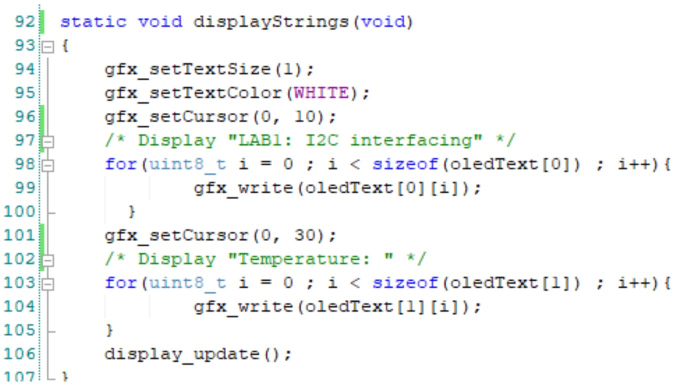
```
/* @brief Include custom libraries for PPG EduKit */
#include "utils.h"
#include "oled_driver.h"
#include "MAX30205.h"
```

- Declare the global variables and function prototypes.



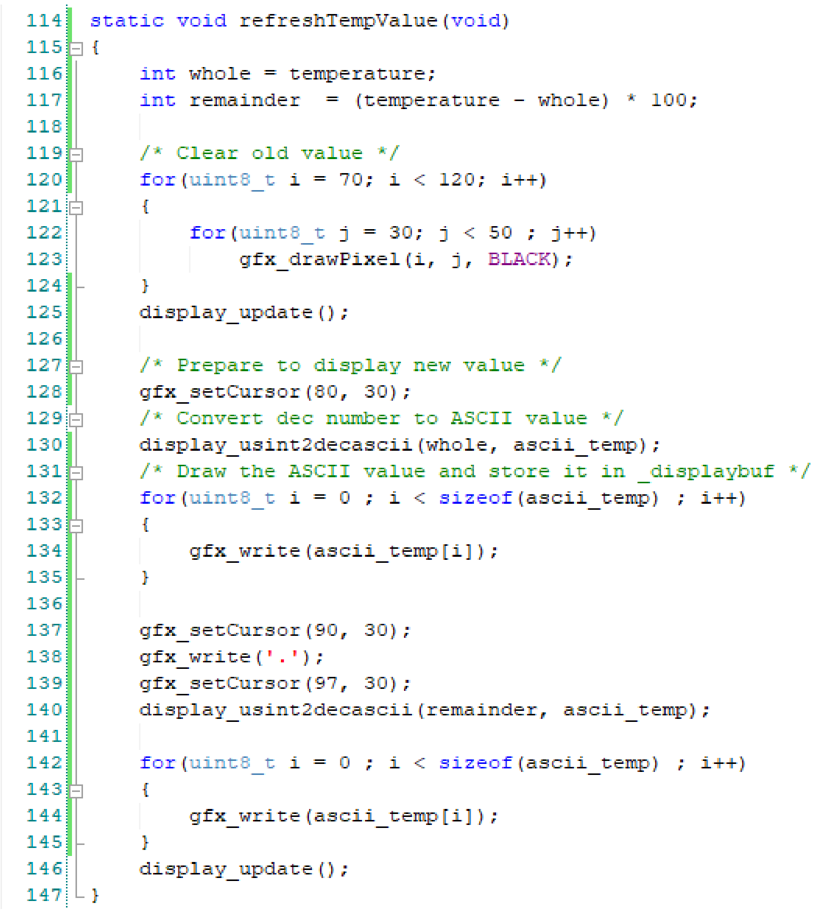
```
/* @brief OLED custom text */
char oledText[][30] = {"LAB1: I2C interfacing", "Temperature: "};
/* @brief Buffer that stores the ASCII values of HR */
char ascii_temp[6];

/* @brief Stores body temperature value */
float temperature;
float lastTemperature;

/*===========================================================================================
 *                                     LOCAL FUNCTION PROTOTYPES
 *  ========================================================================================*/

static void displayStrings(void);
static void refreshTempValue(void);
```

- To display the strings defined in the **oledText** array, **displayStrings** function have to be implemented.



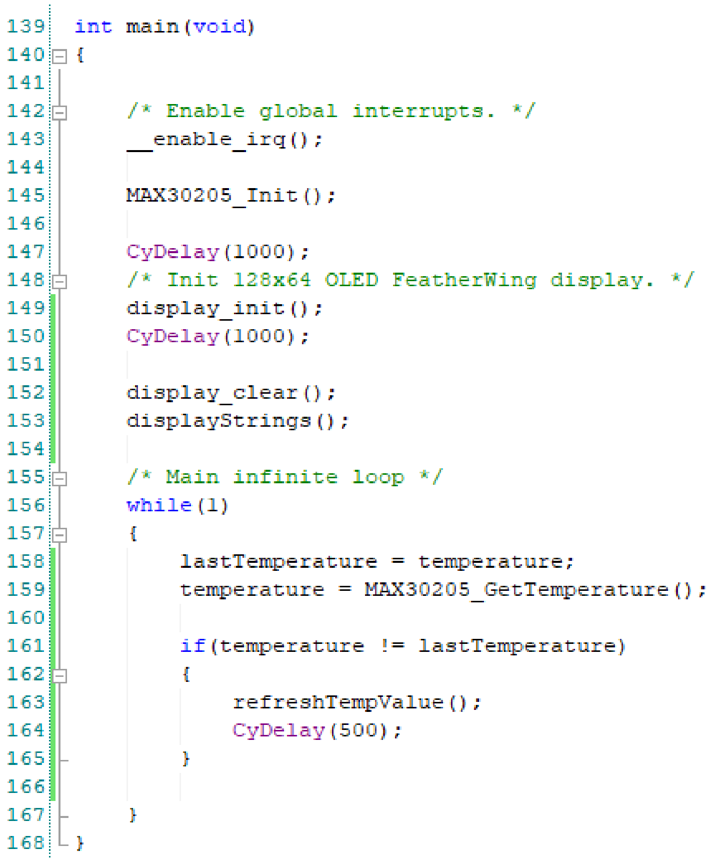
```c
static void displayStrings(void)
{
    gfx_setTextSize(1);
    gfx_setTextColor(WHITE);
    gfx_setCursor(0, 10);
    /* Display "LAB1: I2C interfacing" */
    for(uint8_t i = 0 ; i < sizeof(oledText[0]) ; i++){
            gfx_write(oledText[0][i]);
     }
    gfx_setCursor(0, 30);
    /* Display "Temperature: " */
    for(uint8_t i = 0 ; i < sizeof(oledText[1]) ; i++){
            gfx_write(oledText[1][i]);
    }
    display_update();
}
```

- Each iteration the temperature value has to be updated, while the old value have to be cleared. To achieve this, **refreshTempValue** function have to be implemented.



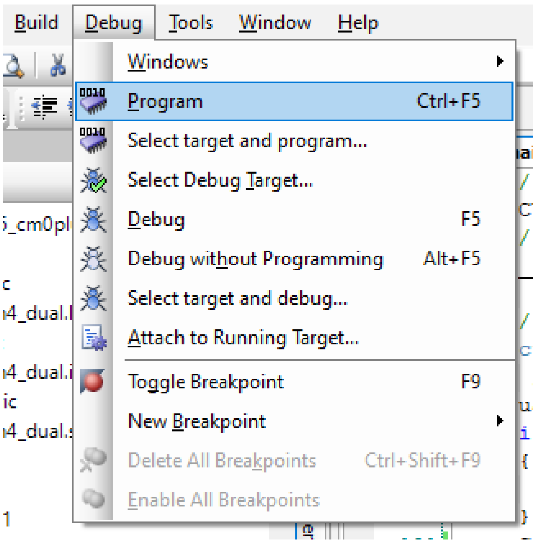
```c
static void refreshTempValue(void)
{
    int whole = temperature;
    int remainder  = (temperature - whole) * 100;

    /* Clear old value */
    for(uint8_t i = 70; i < 120; i++)
    {
        for(uint8_t j = 30; j < 50 ; j++)
            gfx_drawPixel(i, j, BLACK);
    }
    display_update();

    /* Prepare to display new value */
    gfx_setCursor(80, 30);
    /* Convert dec number to ASCII value */
    display_usint2decascii(whole, ascii_temp);
    /* Draw the ASCII value and store it in _displaybuf */
    for(uint8_t i = 0 ; i < sizeof(ascii_temp) ; i++)
    {
        gfx_write(ascii_temp[i]);
    }

    gfx_setCursor(90, 30);
    gfx_write('.');
    gfx_setCursor(97, 30);
    display_usint2decascii(remainder, ascii_temp);

    for(uint8_t i = 0 ; i < sizeof(ascii_temp) ; i++)
    {
        gfx_write(ascii_temp[i]);
    }
    display_update();
}
```

- The main function starts with MAX30205 initialization and with OLED display initialization. Then the predefined strings are displayed. In the while loop, the temperature is read and updated only when the value changes.

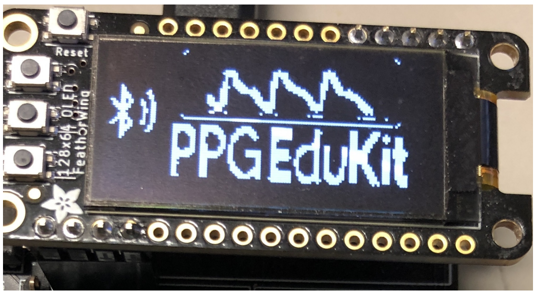


```
int main(void)
{

    /* Enable global interrupts. */
    __enable_irq();

    MAX30205_Init();

    CyDelay(1000);
    /* Init 128x64 OLED FeatherWing display. */
    display_init();
    CyDelay(1000);

    display_clear();
    displayStrings();

    /* Main infinite loop */
    while(1)
    {
        lastTemperature = temperature;
        temperature = MAX30205_GetTemperature();

        if(temperature != lastTemperature)
        {
            refreshTempValue();
            CyDelay(500);
        }

    }
}
```

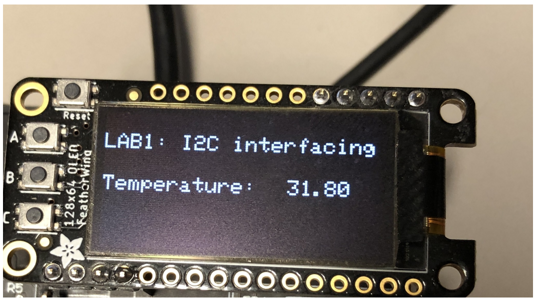
- Build the project and program the target.


Build
Debug
Tools
Window
Help
Windows
Program Ctrl+F5
Select target and program...
Select Debug Target...
Debug F5
Debug without Programming Alt+F5
Select target and debug...
Attach to Running Target...
Toggle Breakpoint F9
New Breakpoint
Delete All Breakpoints Ctrl+Shift+F9
Enable All Breakpoints

- If the display initialization is correct, the following image should be seen for a period of 1 second.



- Cover the sensor's surface with your finger. Then the temperature should be displayed and updated every time the value changes.



```
LAB1: I2C interfacing
Temperature:   31.80
```

References

[1] https://www.displayfuture.com/Display/datasheet/controller/SH1107.pdf

[2] https://datasheets.maximintegrated.com/en/ds/MAX30205.pdf

[3] Cypress, ”PSoC 6 MCU: CY8C63x6, CY8C63x7 Datasheet”, https://www.cypress.com/file/385921/, Nov. 2020

[4] Cypress, ”PSoC 6 MCU Code Examples with PSoC Creator”, https://www.cypress.com/documentation/code-examples/psoc-6-mcu-code-examples-psoc-creator, Mar. 2020

[5] https://www.cypress.com/file/137441/download

[6] https://phoenixnap.com/kb/install-pip-windows