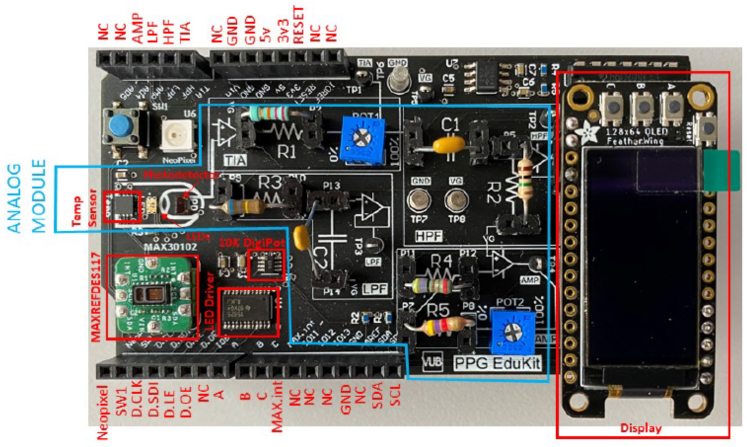
Lab 1: Interface I2C devices in PPG EduKit platform with Arduino.

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 as a shield for any Arduino board as long as the microcontroller runs with a logic level voltage of 3V3. **Do not use the PPG EduKit with microcontrollers that have a logic level voltage of 5V, like the Arduino UNO.**

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

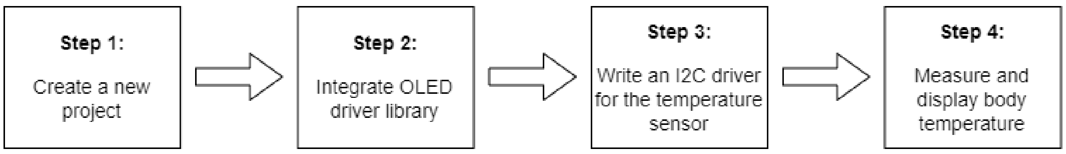
After completing this lab, you will be able to:

- 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 Arduino IDE.
- Go to File → New.
- After the sketch is created, save the sketch (File → Save As) as **Measure_temperature**.
- Copy the PPG EduKit library (PPG_EduKit.cpp and PPG_EduKit.h) in **Measure_temperature** directory.

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 it is integrated inside PPG_EduKit class.

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.

- Before integrating the OLED library inside an Arduino project, it is needed to import the PPG_EduKit library and to declare an instance of the class.

```
#include "PPG_EduKit.h"

PPG_EduKit PPG_Shield;
```

- Inside **setup** function, call the **begin** function. The function checks the peripheral list structure to determine which components to be initialized. Set the **oledDisplay** member as **ENABLE_PERIPHERAL**.

```
void setup() {

  PPG_EK_Peripherals periphList = {
    .oledDisplay = ENABLE_PERIPHERAL,
    .neoPixel = DISABLE_PERIPHERAL,
    .tempSensor = DISABLE_PERIPHERAL,
    .ppgSensor = DISABLE_PERIPHERAL,
    .read_TIA = DISABLE_PERIPHERAL,
    .read_HPF = DISABLE_PERIPHERAL,
    .read_LPF = DISABLE_PERIPHERAL,
    .read_AMP = DISABLE_PERIPHERAL
  };
  PPG_Shield.begin(&periphList, samplingRate);
}
```

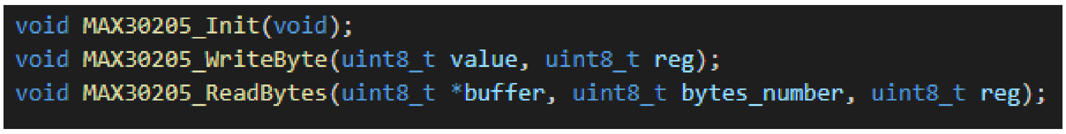
- To access the OLED functions one has to declare a reference to an object of type **Adafruit_SH110X** and to use the **getHandler** function.

```
Adafruit_SH110X &display_reference = PPG_Shield.getHandler_OLED();
display_reference.clearDisplay();
```

3 Interface MAX30205 temperature sensor

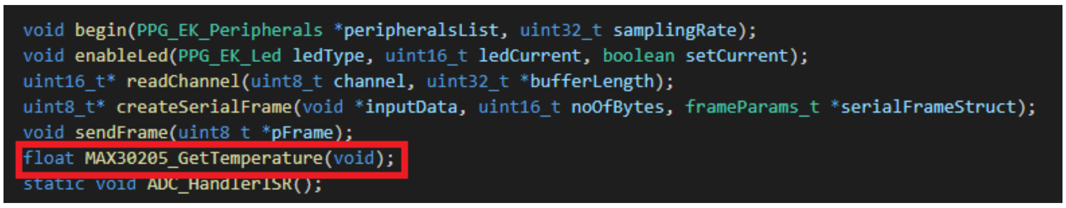
The body temperature sensor can be interfaced through an I2C-compatible, 2-wire serial interface. The I2C serial interface accepts standard write byte, read byte, send byte, and receive byte commands to read the temperature data and configure the behavior of the opendrain overtemperature shutdown output. 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.

- 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 class (**PPG_EduKit.h**) as private functions.



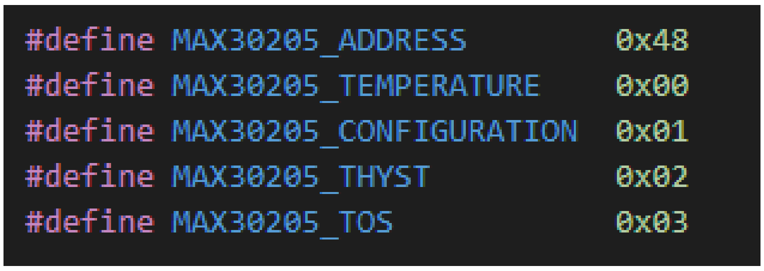
```
void MAX30205_Init(void);
void MAX30205_WriteByte(uint8_t value, uint8_t reg);
void MAX30205_ReadBytes(uint8_t *buffer, uint8_t bytes_number, uint8_t reg);
```

- Declare the **getTemperature** function inside the class (**PPG_EduKit.h**) as public function.



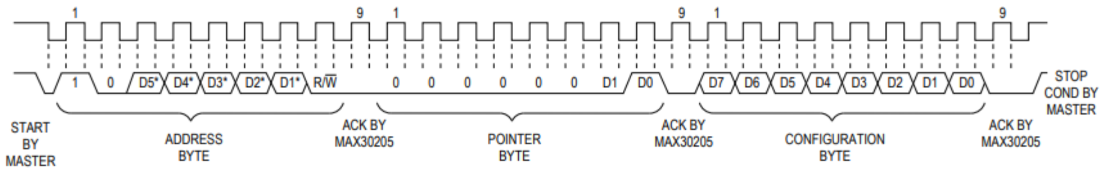
```
void begin(PPG_EK_Peripherals *peripheralsList, uint32_t samplingRate);
void enableLed(PPG_EK_Led ledType, uint16_t ledCurrent, boolean setCurrent);
uint16_t* readChannel(uint8_t channel, uint32_t *bufferLength);
uint8_t* createSerialFrame(void *inputData, uint16_t noOfBytes, frameParams_t *serialFrameStruct);
void sendFrame(uint8_t *pFrame);
float MAX30205_GetTemperature(void);
static void ADC_HandlerISR();
```

- 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.
- Declare the I2C address of the device and the registers addresses (see datasheet [2]).



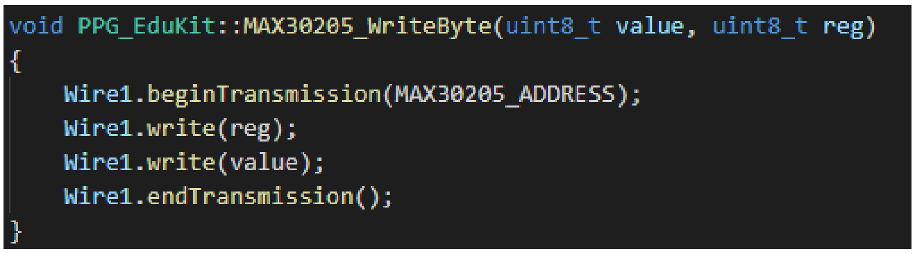
```
#define MAX30205_ADDRESS        0x48
#define MAX30205_TEMPERATURE    0x00
#define MAX30205_CONFIGURATION  0x01
#define MAX30205_THYST          0x02
#define MAX30205_TOS            0x03
```

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



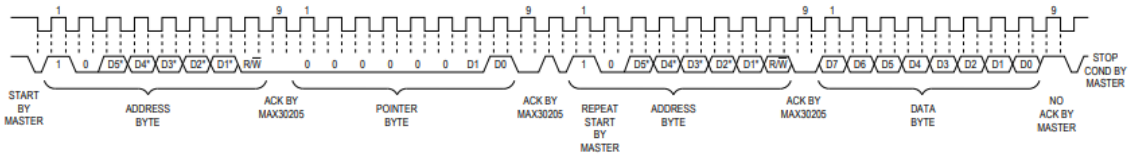
(a) CONFIGURATION REGISTER WRITE.

- 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.
- Go to **PPG_EduKit.cpp** and implement the **WriteByte** function. Depending on what type of Arduino board is used, the **Wire1** library should be replaced by **Wire** library.



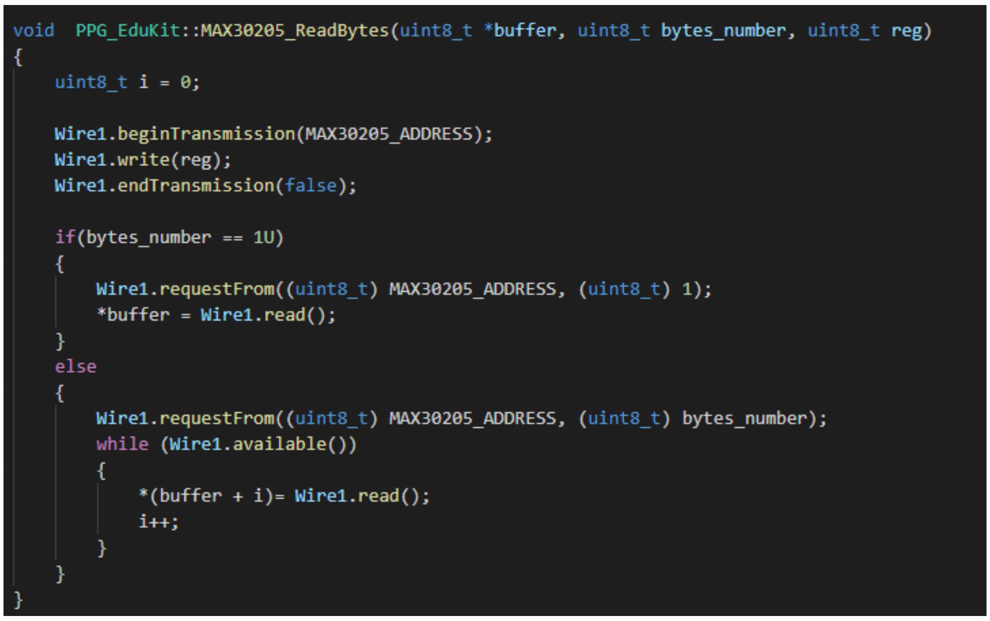
```
void PPG_EduKit::MAX30205_WriteByte(uint8_t value, uint8_t reg)
{
    Wire1.beginTransmission(MAX30205_ADDRESS);
    Wire1.write(reg);
    Wire1.write(value);
    Wire1.endTransmission();
}
```

- 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.



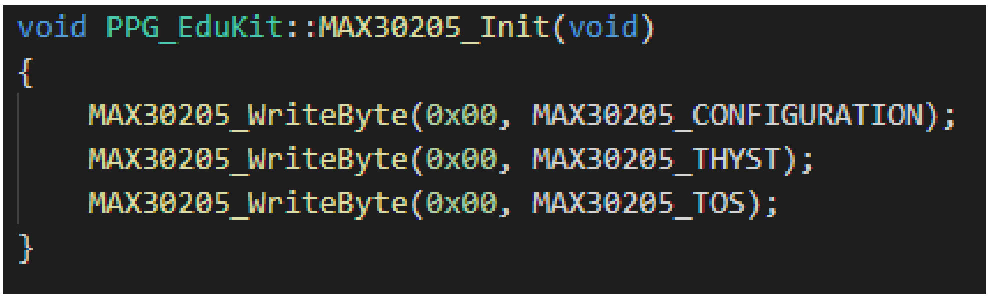
(a) TYPICAL POINTER SET FOLLOWED BY IMMEDIATE READ FROM CONFIGURATION REGISTER.

- 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.



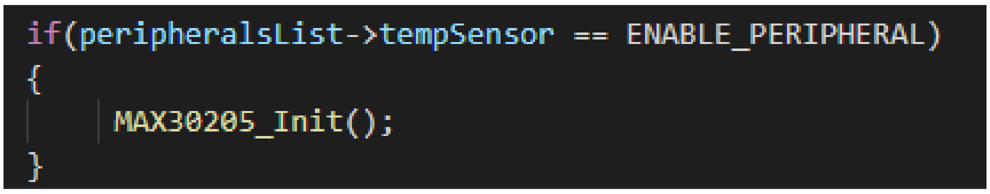
```
void  PPG_EduKit::MAX30205_ReadBytes(uint8_t *buffer, uint8_t bytes_number, uint8_t reg)
{
    uint8_t i = 0;

    Wire1.beginTransmission(MAX30205_ADDRESS);
    Wire1.write(reg);
    Wire1.endTransmission(false);

    if(bytes_number == 1U)
    {
        Wire1.requestFrom((uint8_t) MAX30205_ADDRESS, (uint8_t) 1);
        *buffer = Wire1.read();
    }
    else
    {
        Wire1.requestFrom((uint8_t) MAX30205_ADDRESS, (uint8_t) bytes_number);
        while (Wire1.available())
        {
            *(buffer + i)= Wire1.read();
            i++;
        }
    }
}
```

- Before calling any of the functions above, the initialization function have to be defined.



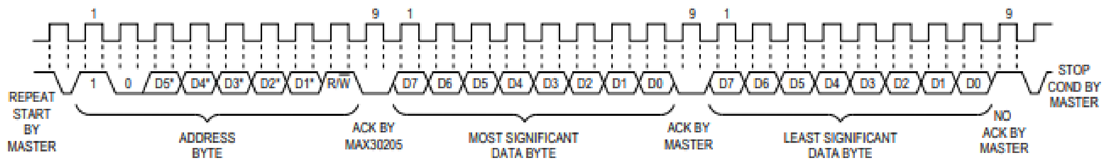
```
void PPG_EduKit::MAX30205_Init(void)
{
    MAX30205_WriteByte(0x00, MAX30205_CONFIGURATION);
    MAX30205_WriteByte(0x00, MAX30205_THYST);
    MAX30205_WriteByte(0x00, MAX30205_TOS);
}
```

- Call the function inside **PPG_EduKit::begin** function.

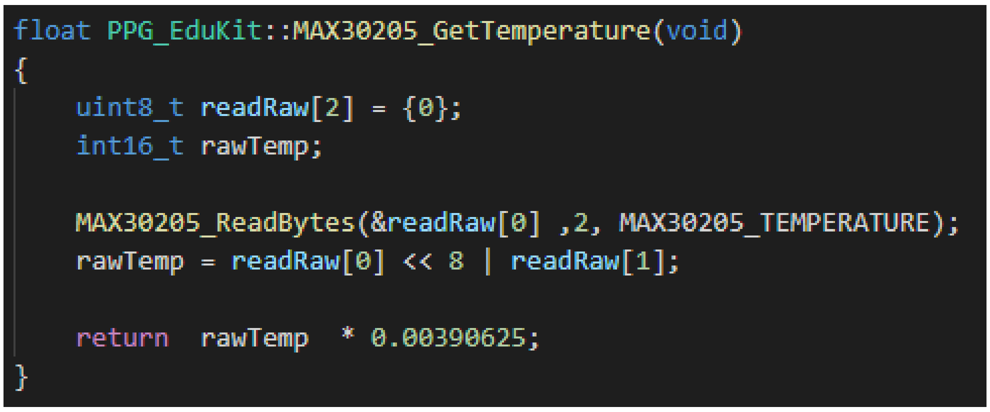


```
if(peripheralsList->tempSensor == ENABLE_PERIPHERAL)
{
    MAX30205_Init();
}
```

- 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).

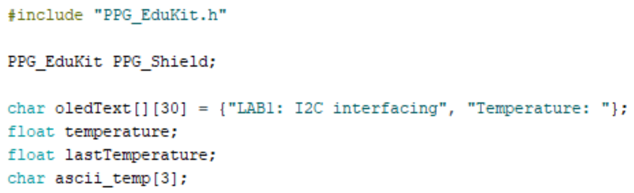


```
float PPG_EduKit::MAX30205_GetTemperature(void)
{
    uint8_t readRaw[2] = {0};
    int16_t rawTemp;

    MAX30205_ReadBytes(&readRaw[0] ,2, MAX30205_TEMPERATURE);
    rawTemp = readRaw[0] << 8 | readRaw[1];

    return  rawTemp  * 0.00390625;
}
```

4 Create the main program

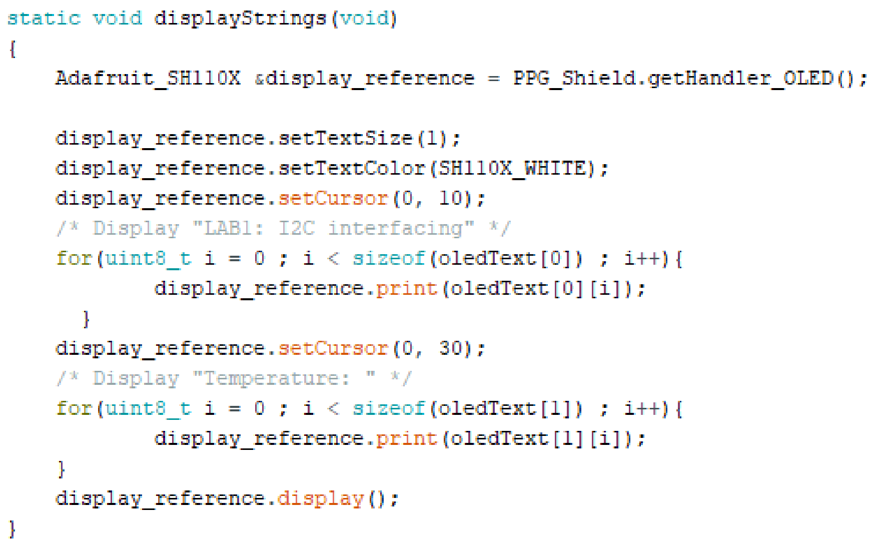
The main program should merge together all the libraries in such a way to reach the goal of displaying the finger temperature on the OLED display.

- Start by including the **PPG_EduKit** library and declare the global variables.



```
#include "PPG_EduKit.h"

PPG_EduKit PPG_Shield;

char oledText[][30] = {"LAB1: I2C interfacing", "Temperature: "};
float temperature;
float lastTemperature;
char ascii_temp[3];
```

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



```
static void displayStrings(void)
{
    Adafruit_SH110X &display_reference = PPG_Shield.getHandler_OLED();

    display_reference.setTextSize(1);
    display_reference.setTextColor(SH110X_WHITE);
    display_reference.setCursor(0, 10);
    /* Display "LAB1: I2C interfacing" */
    for(uint8_t i = 0 ; i < sizeof(oledText[0]) ; i++){
            display_reference.print(oledText[0][i]);
      }
    display_reference.setCursor(0, 30);
    /* Display "Temperature: " */
    for(uint8_t i = 0 ; i < sizeof(oledText[1]) ; i++){
            display_reference.print(oledText[1][i]);
    }
    display_reference.display();
}
```

- 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. The temperature has to be split into the fractional part and the integer part.

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

    Adafruit_SH110X &display_reference = PPG_Shield.getHandler_OLED();
    /* Clear old value */
    for(uint8_t i = 70; i < 120; i++)
    {
        for(uint8_t j = 30; j < 50 ; j++)
            display_reference.drawPixel(i, j, SH110X_BLACK);
    }
    display_reference.display();
    /* Prepare to display new value */
    display_reference.setCursor(80, 30);
    /* Convert dec number to ASCII value */
    PPG_Shield.int2acii(temperature, ascii_temp);
    /* Draw the ASCII value and store it in _displaybuf */
    for(uint8_t i = 0 ; i < sizeof(ascii_temp) ; i++){
        display_reference.print(ascii_temp[i]);
    }
    display_reference.setCursor(96, 30);
    display_reference.print(".");
     /* Prepare to display new value */
    display_reference.setCursor(97, 30);
    /* Convert dec number to ASCII value */
    PPG_Shield.int2acii(remainder, ascii_temp);
    /* Draw the ASCII value and store it in _displaybuf */
    for(uint8_t i = 0 ; i < sizeof(ascii_temp) ; i++){
        display_reference.print(ascii_temp[i]);
    }
    display_reference.display();
}
```

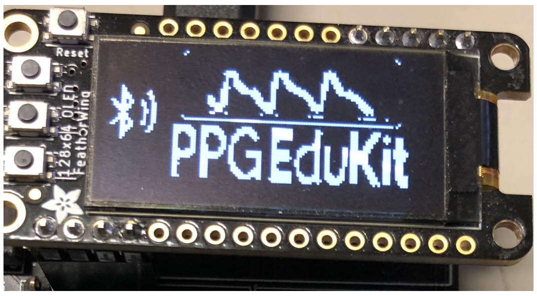
- The setup function should initialize the shield by calling the **begin** method. The sampling rate is set to 0 since no ADC reading has to be performed. Then, the display is cleared and the predefined strings are displayed.

```
void setup() {

  PPG_EK_Peripherals periphList = {
    .oledDisplay = ENABLE_PERIPHERAL,
    .neoPixel = DISABLE_PERIPHERAL,
    .tempSensor = ENABLE_PERIPHERAL,
    .ppgSensor = DISABLE_PERIPHERAL,
    .read_TIA = DISABLE_PERIPHERAL,
    .read_HPF = DISABLE_PERIPHERAL,
    .read_LPF = DISABLE_PERIPHERAL,
    .read_AMP = DISABLE_PERIPHERAL
  };

  delay(1000);
  PPG_Shield.begin(&periphList, 0);
  delay(1000);

  Adafruit_SH110X &display_reference = PPG_Shield.getHandler_OLED();
  display_reference.clearDisplay();

  displayStrings();
}
```

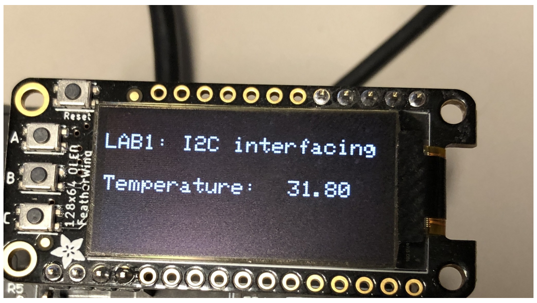
- The **loop** function reads the temperature from MAX30205 and refreshes the values on the display every time the temperature changes.

```
void loop() {

  lastTemperature = temperature;
  temperature = PPG_Shield.MAX30205_GetTemperature();

  if(temperature != lastTemperature)
  {
      refreshTempValue();
      Serial.println(temperature);
      delay(500);
  }
}
```

- Build the project and program the target.
- 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