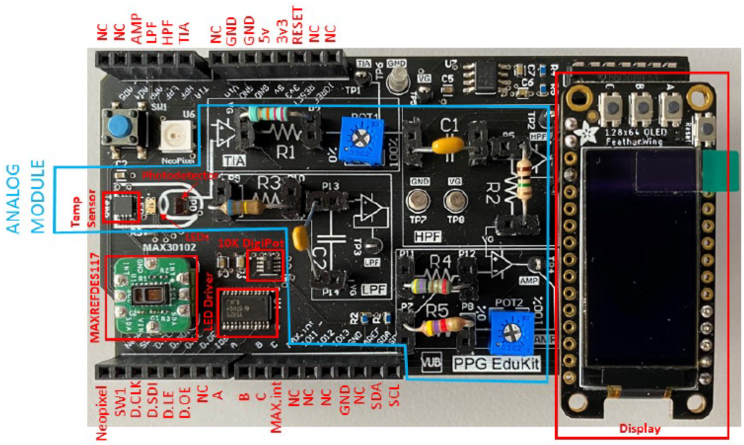
ETRO ELECTRONICS & INFORMATICS

Lab 6: BLE. Compute HR and control the wavelength of the measurement

Introduction

This lab helps the user to acquire the PPG signal using two different sensor setups: using the analog conditioning stage which uses one photodetector and three different wavelengths (green, red and IR) and using the commercial MAXREFDES117 PPG module that includes the MAX30102 sensor with only two wavelengths (red and IR). The heart rate value will be computed for both sensors and printed out on the display. The wavelength for the custom solution can be controlled over BLE with the help of a phone.

The PPG EduKit platform is shown below. The module can be used with the CY8CPROTO-063-BLE board using the bridge adaptor provided. The analog PPG module includes one RGB LED, a photodetector, a transimpedance amplifier, a high pass filter, a low pass filter and an amplification stage. In this lab the PPG signal is acquired from the last amplification stage, thus the board has to be configured with the proper component values.



Objectives

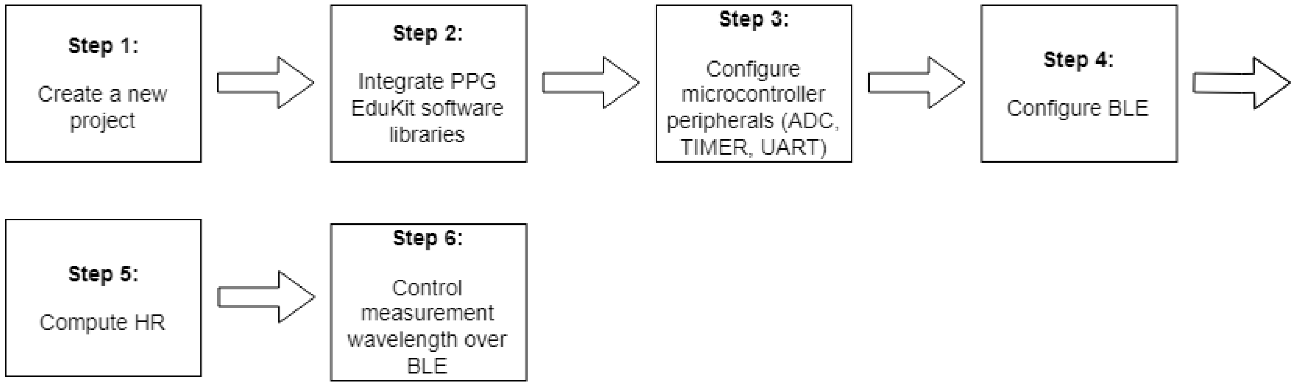
After completing this lab, you will be able to:

- Read data from MAX30102 sensor and custom PPG sensor
- Learn how to compute the heart rate
- Configure the BLE in PSOC Creator
- Communicate with the board over BLE

Procedure

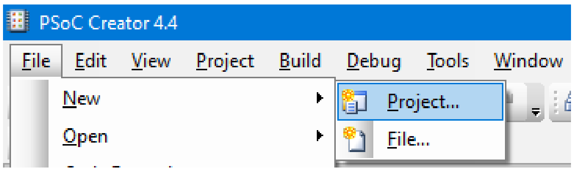
This lab is separated into steps that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

The lab includes 6 primary steps: create a project using PSOC Creator for CY8CPROTO-063-BLE board, integrate the software libraries in the newly created project (LED driver, digital potentiometer, MAX30102 sensor), configure the peripherals in order to read the PPG signal, configure BLE and interrupts, compute HR and compare the sensor module with the analog module, and finally control the wavelength of the measurement over BLE.

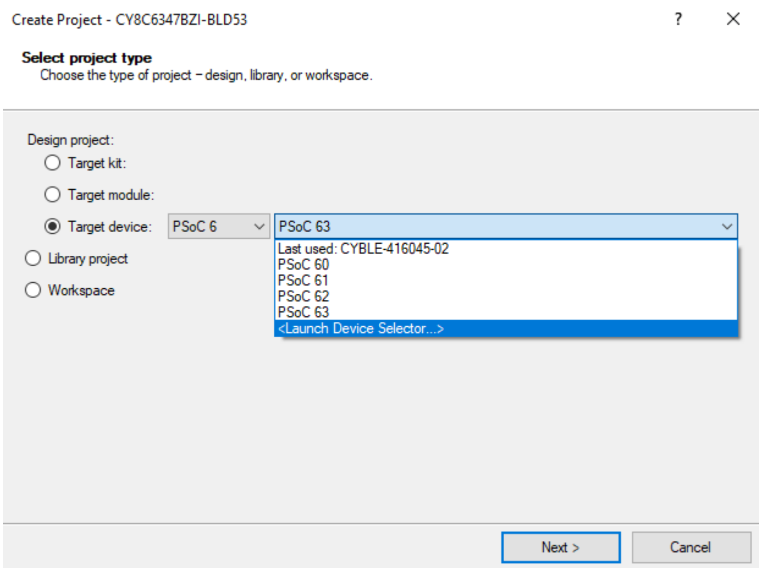


1 Creation of the Project

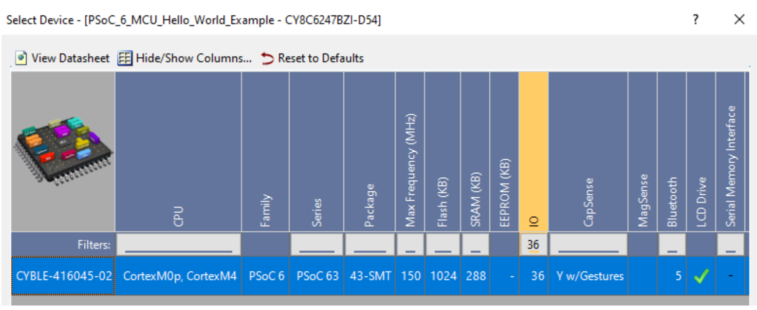
- Open PSoC Creator
- Go to File → New → Project



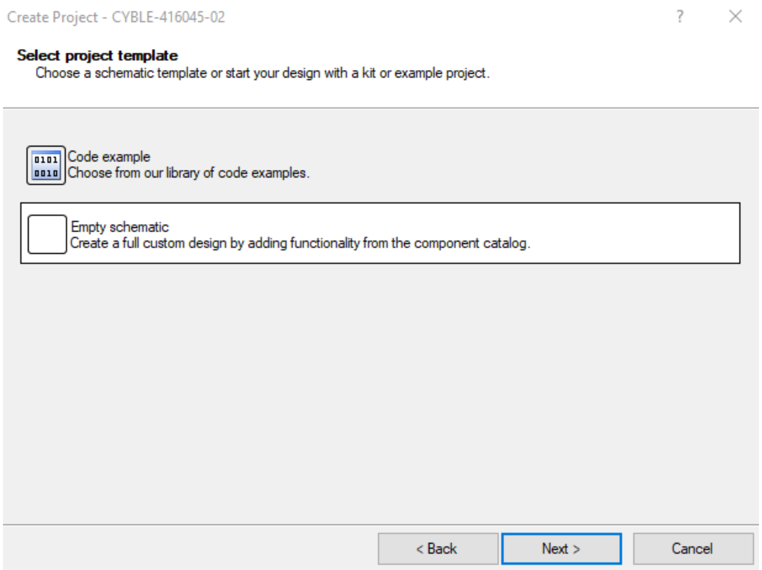
- Select the target device → PSOC6 → <Launch Device Selector>



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



- Select "Empty schematic" and click "Next"



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

2 Integrate PPG EduKit software libraries

The next step is to integrate the software drivers that are provided in the PPG EduKit package. As seen in the application note, some external components need to be interfaced using SPI or I2C. Libraries are provided to speed up and to facilitate the development of PPG related applications, such that the user can focus on PPG signal acquisition or other types of algorithms.

In order to compute the HR and control the wavelength over BLE, the user has to integrate the following libraries: TLC5925 (LED driver library), AD5273 (digital potentiometer library), custom PPG, HR algorithm, OLED display, serial frame (UART), MAX30102 module, milliseconds and utils library. Each library consists of a source file (.c) and a header file (.h) and can be found in PPG EduKit package.

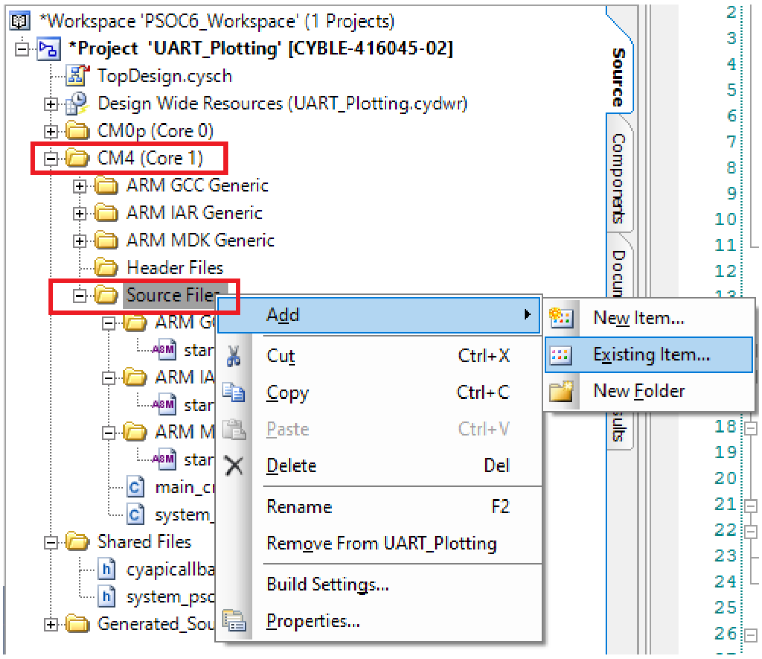
Utils library

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

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

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

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

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

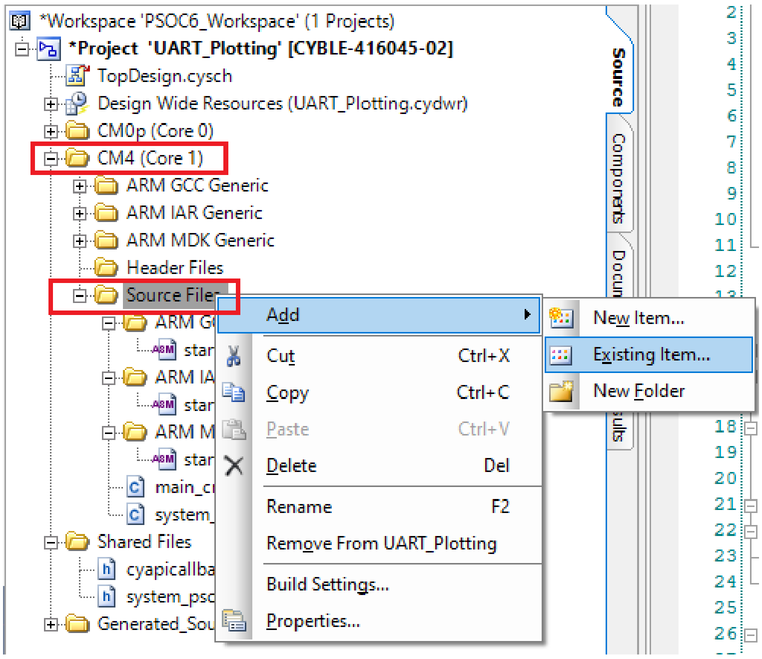


- Repeat the process for the header file (utils.h). Import the file in CM4 (Core1) → Header Files

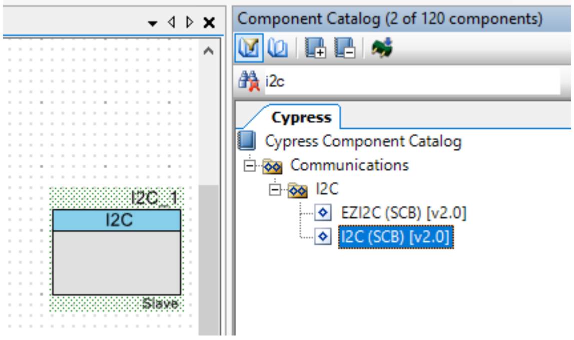
AD5273 driver

AD5273 is a 64-position, one-time programmable (OTP) digital potentiometer that employs fuse link technology to achieve permanent program setting and can be configured through an I2C-compatiblle interface.

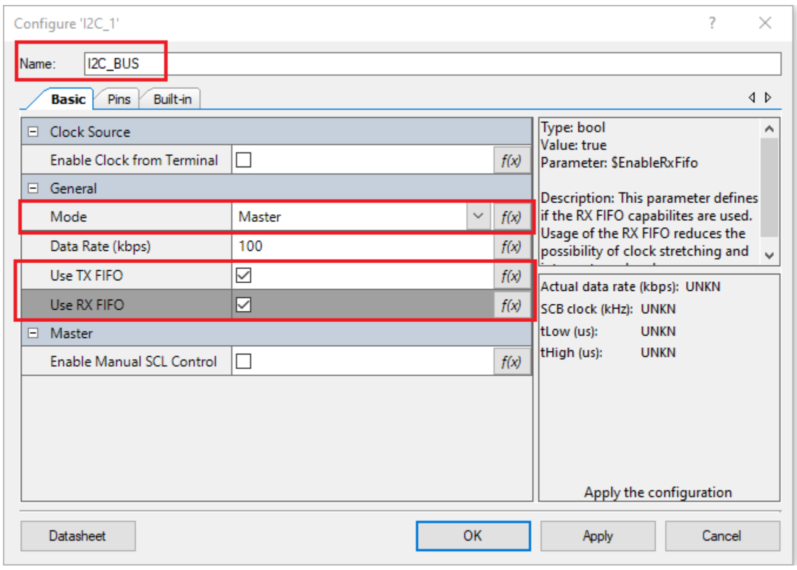
- Import the source file (AD5273.c) in CM4 (Core1) → Source Files



- Repeat the process for the header file (AD5273.h). Import the file 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.

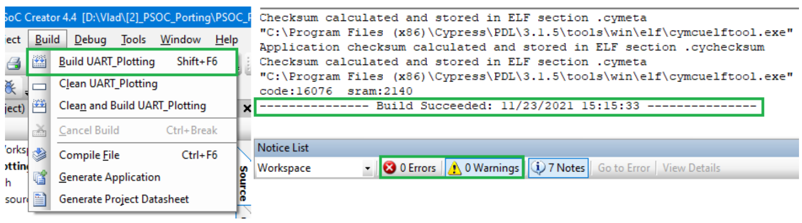


- Configure the I2C component by double click on it. Rename the component as **I2C_BUS**, set the mode as **master** and enable **TX/RX FIFO** buffers.



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

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

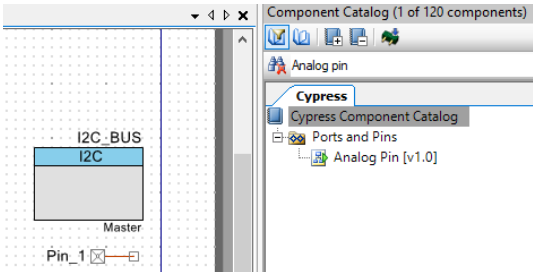


- Take a look at the application programming interface (AD5273.c) and try to understand the functions of the driver.

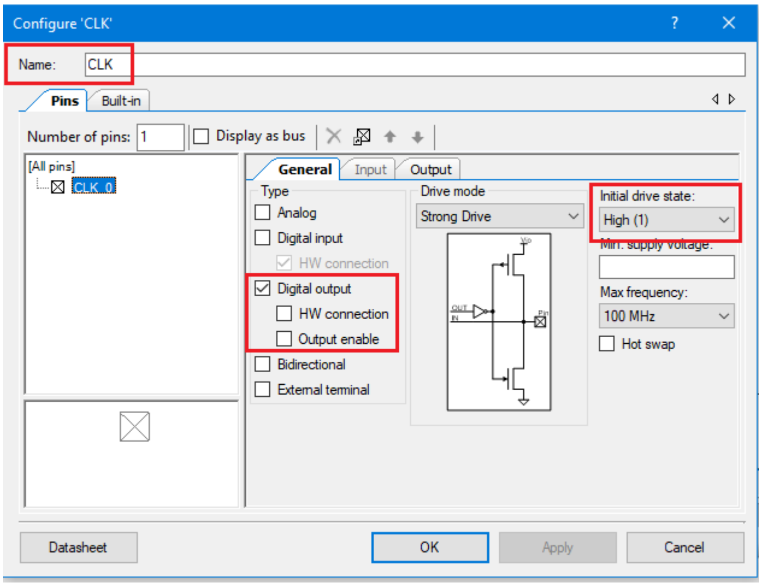
TLC5925 driver

TLC5925 low-power 16-channel constant-current LED sink drivers to contain a 16-bit shift register and data latches, leading to converted serial input data into a parallel output format. The serial data is transferred into the device via **SDI** line at every rising edge of the **CLK** line. **LE** line latch the serial data in the shift register to the output latch, and the **OE** line enables the output drivers to sink current.

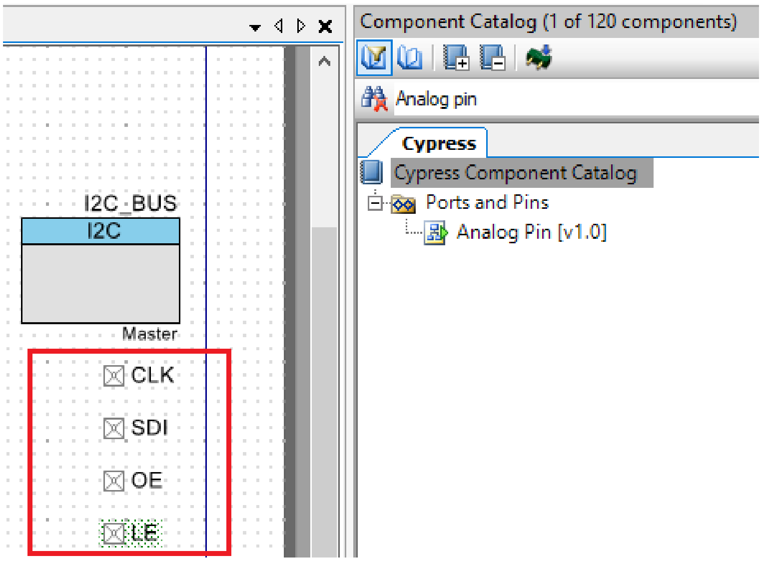
- Import the source file (TLC5925.c) in CM4 (Core1) → Source Files
- Repeat the process for the header file (TLC5925.h). Import the file in CM4 (Core1) → Header Files
- Go to the **TopDesign** schematic. In the component catalog (right panel at the right of the screen), write **Analog pin** and drag and drop the component into the schematic.



- Configure the Pin component by double click on it.



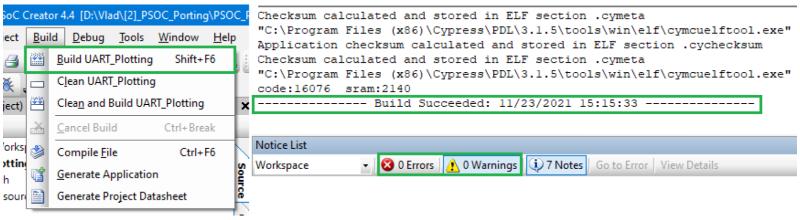
- Repeat the step for SDI, OE and LE pins.



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

	CLK	P12[6]	34	☑
	LE	P12[7]	35	☑
	OE	P7[2]	22	☑
	SDI	P0[5]	2	☑

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

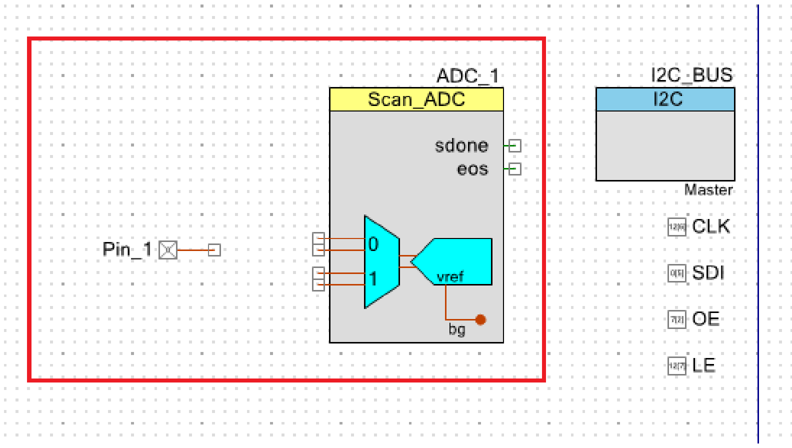


- Take a look at the application programming interface (TLC5925.c) and try to understand functions of the driver.

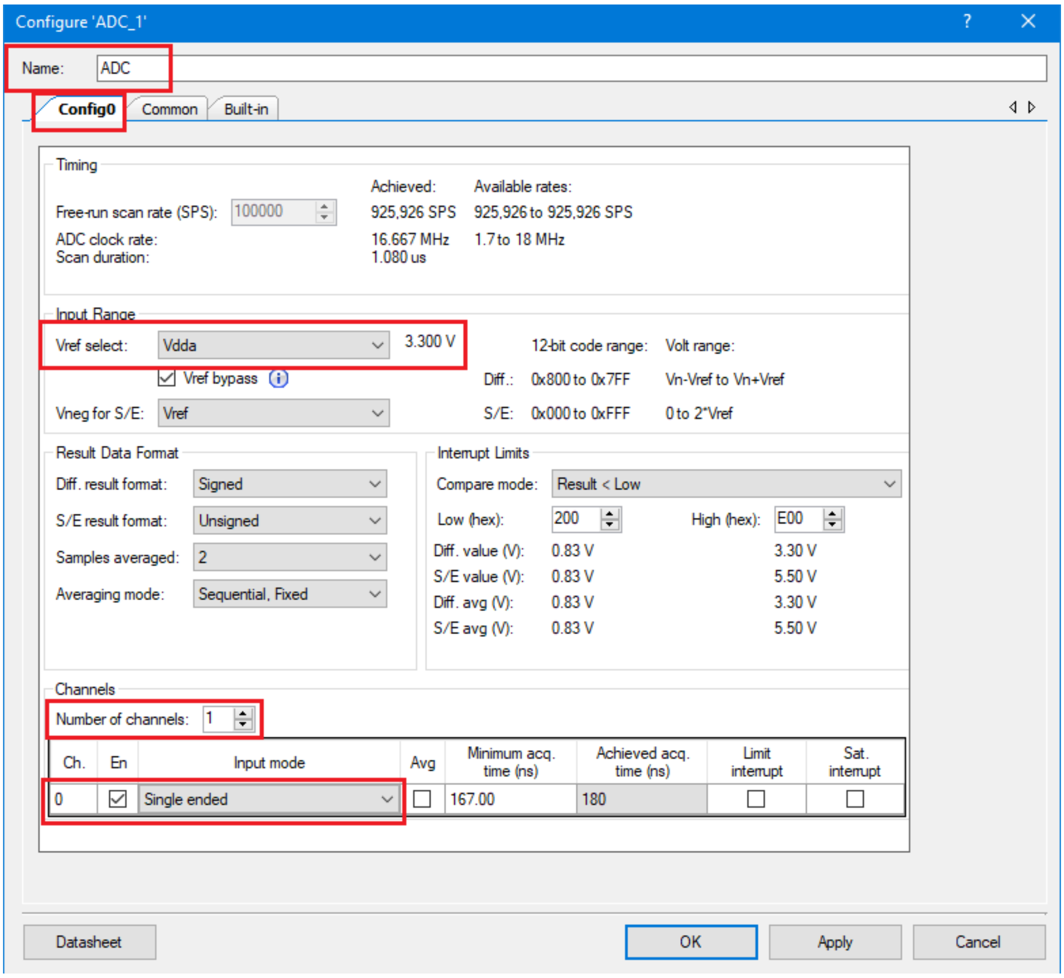
Custom PPG driver

The Custom PPG library is intended to include the user defined algorithms such as HR, SpO2 or digital filters. In this lab, the library includes only the ADC reading routine using an timer triggered ISR.

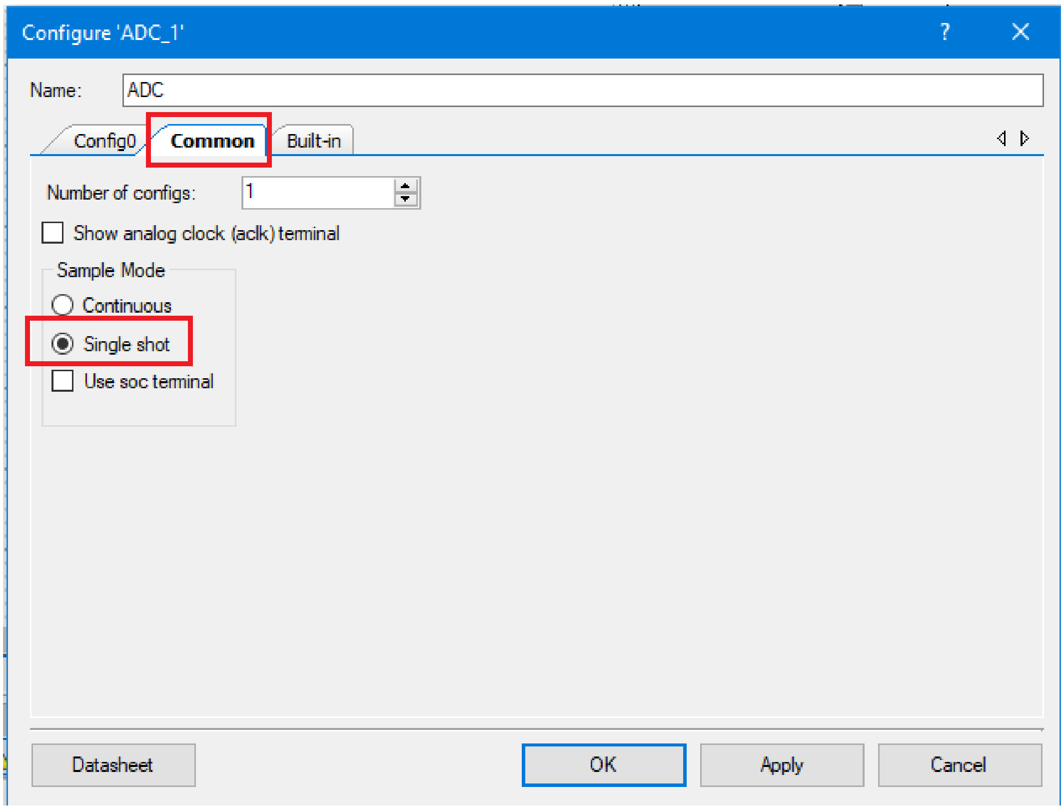
- Go to the **TopDesign** schematic. In the component catalog (right panel at the right of the screen), write **ADC** and drag and drop the component into the schematic. Search for **Analog pin** and drag and drop the component into the schematic.



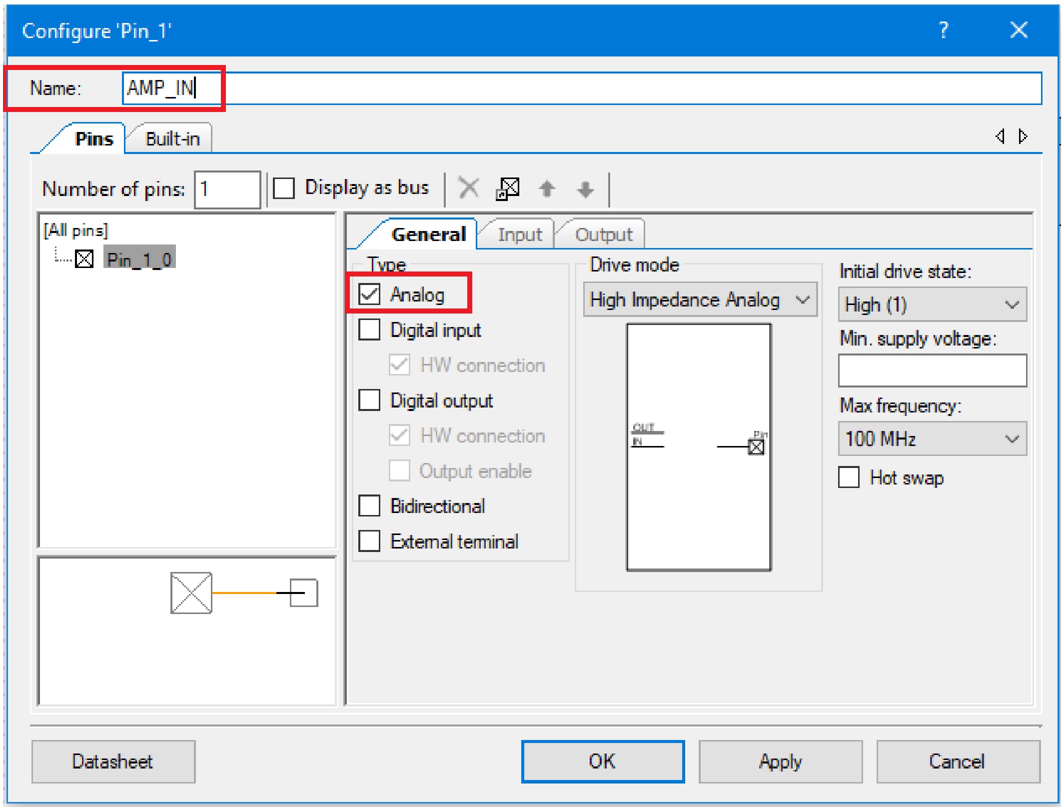
- Configure the ADC component by double click on it.



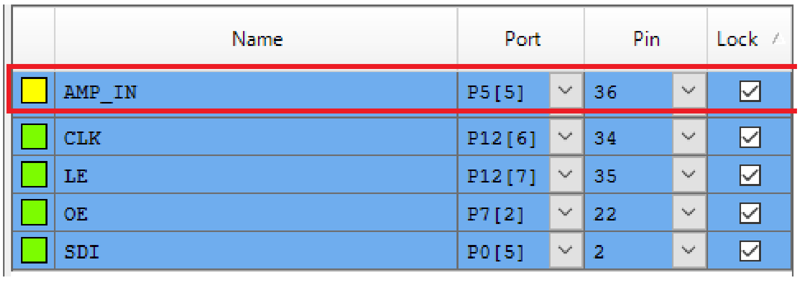
- Configure the ADC sample mode as single shot mode.



- Configure the Pin component by double click on it.

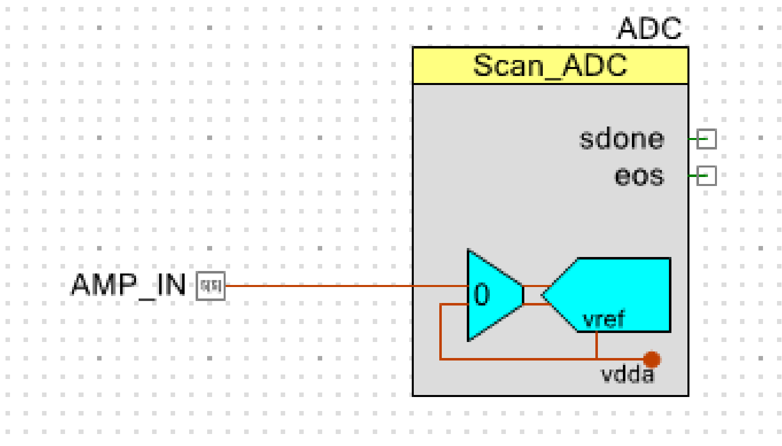


- Go to Design Wide Resources → Pins and assign the **AMP_IN** pin as follows:

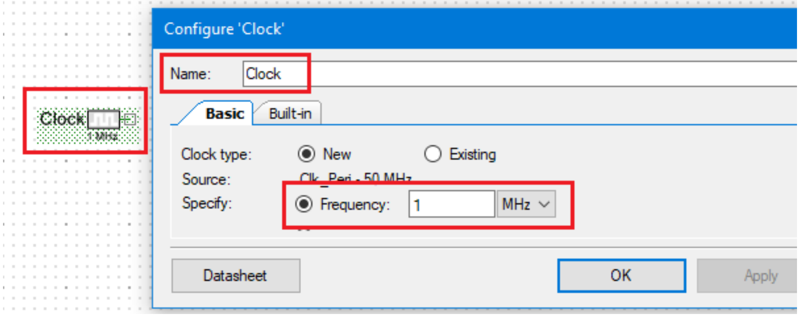


	Name	Port	Pin	Lock
	AMP_IN	P5[5]	36	☑
	CLK	P12[6]	34	☑
	LE	P12[7]	35	☑
	OE	P7[2]	22	☑
	SDI	P0[5]	2	☑

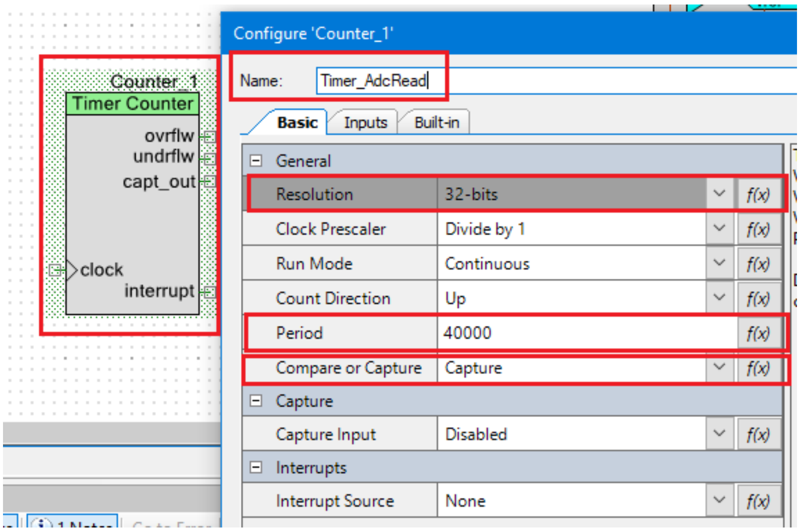
- Connect the analog pin to the ADC (press W to place a wire).



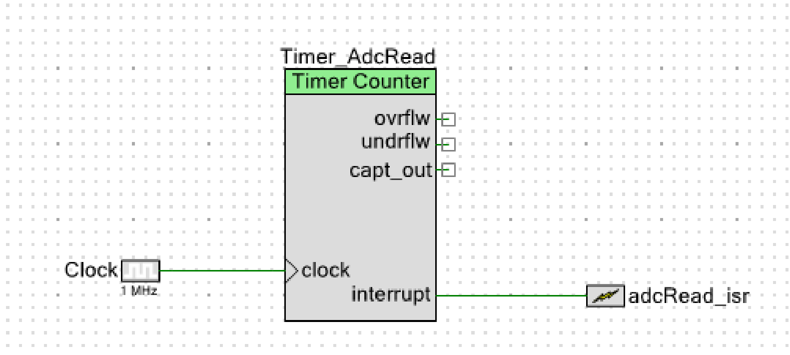
- Search for **Clock** and drag and drop the component into the schematic.
- Configure clock to 1 MHz



- Search for **Timer counter** and drag and drop the component into the schematic.
- Configure the counter.



- Search for **Interrupt** and drag and drop the component into the schematic. Rename the component as **adcRead_isr**.
- Interconnect the components

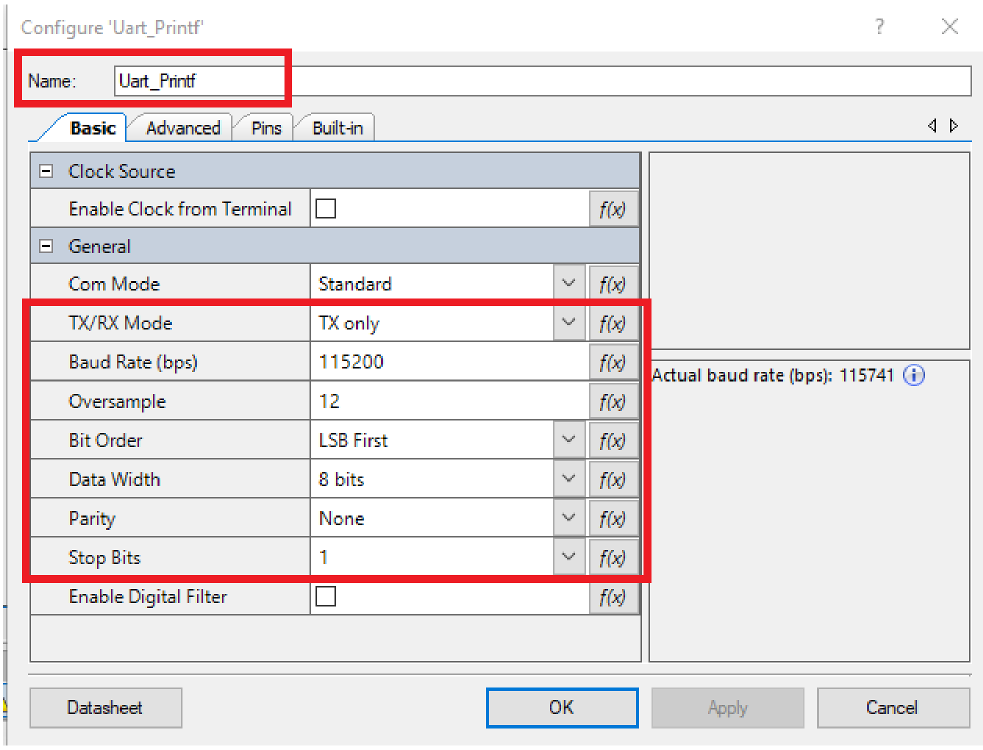


- Import the source file (custom_ppg.c) in CM4 (Core1) → Source Files
- Repeat the process for the header file (custom_ppg.h). Import the file in CM4 (Core1) → Header Files
- Build the project to check if there is any error.
- Take a look at the application programming interface (custom_ppg.c) and try to understand functions of the driver.

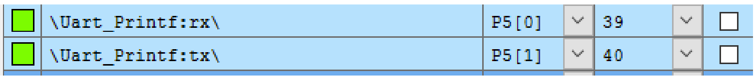
Serial Frame library

This library is a UART wrapper used to interface the PSOC with the PPG EduKit GUI. UART Communication stands for Universal asynchronous receiver-transmitter. It is a dedicated hardware device that performs asynchronous serial communication. It provides features for the configuration of data format and transmission speeds at different baud rates.

- Go to the **TopDesign** schematic. In the component catalog (right panel at the right of the screen), search for **UART** and drag and drop the component into the schematic.
- Configure the component.



- Go to Design Wide Resources → Pins and assign the UART pin as follows:

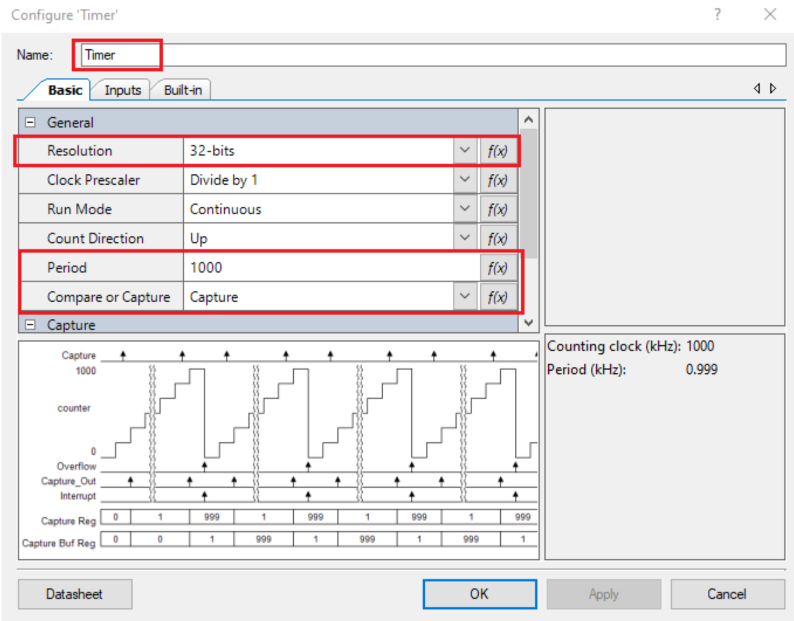


- Import the source file (serial_frame.c) in CM4 (Core1) → Source Files
- Repeat the process for the header file (serial_frame.h). Import the file in CM4 (Core1) → Header Files
- Build the project to check if there is any error.
- Take a look at the application programming interface (serial_frame.c) and try to understand functions of the driver.

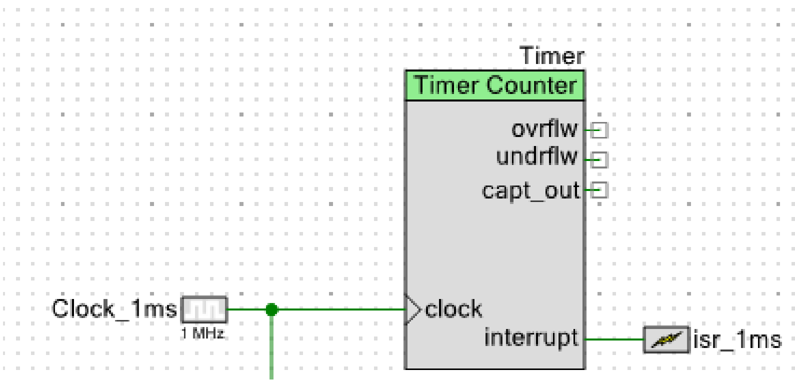
Milliseconds library

Before including the MAX30102 library, the milliseconds library has to be configured and included. As the Arduino function **millis**, we needed such a type of library to be used in PSOC. Porting code from Arduino to PSOC requires sometimes creating new libraries, since the microcontroller access layer is different for both platforms.

- Search for **Timer counter** and drag and drop the component into the schematic.
- Configure the counter. With a clock source of 1MHz, a 1000 period represents 1kHz (1ms).



- Search for **Interrupt** and drag and drop the component into the schematic. Rename the component as **isr_1ms**.
- Interconnect the components, use the same clock source (1MHz).



The **isr_1ms** increments a counter every millisecond. In such a way, the functionality of the **millis** function is achieved. For instance, the sensor can be pooled for a period of time using the newly milliseconds library.

```
/*************************************************************************
 * Function: MAX30102_SafeCheck
 *
 * Description: Pool sensor for new samples
 *************************************************************************/
bool MAX30102_SafeCheck(uint8_t maxTimeToCheck)
{
  uint32_t markTime;

  markTime  = MILLIS_GetValue();

  while(1)
  {
    if(MILLIS_GetValue() - markTime > maxTimeToCheck) return(false);

    if(MAX30102_Check() == true) //We found new data!
      return(true);

    CyDelayUs(1);
  }
}
```

MAX30102 library

The MAX30102 is an integrated pulse oximetry and heart-rate monitor module. It includes internal LEDs, photodetectors, optical elements, and low-noise electronics with ambient light rejection. The MAX30102 provides a complete system solution to ease the design-in process for mobile and wearable devices. The original library is provided by SparkFun as an Arduino library [?]. The library used in this project is a ported version of the original library, since PSOC6 and Arduino platforms are not compatible. The driver interface includes multiple functions and that can be found in **MAX30102.h** file.

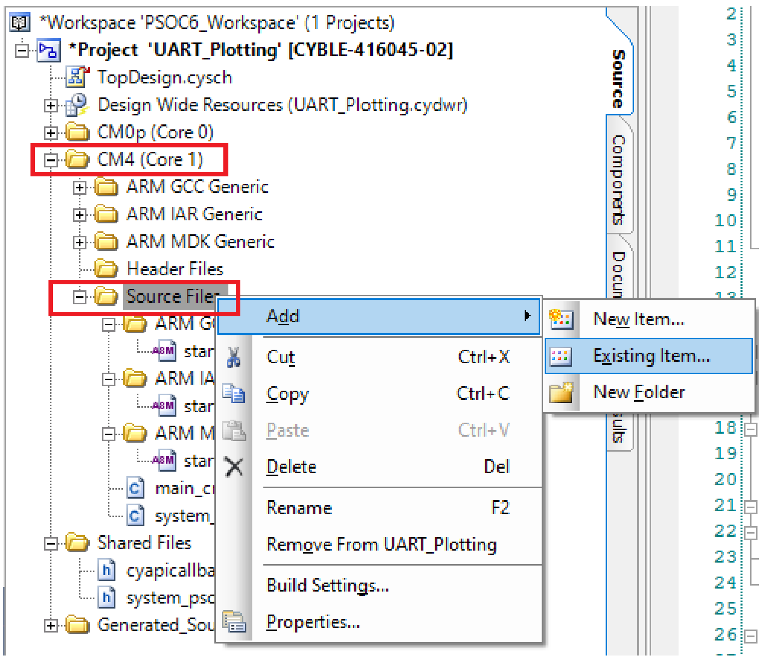
```
void MAX30102_NextSample(void);
void MAX30102_SetFIFOAverage(uint8_t numberOfSamples);
void MAX30102_EnableFIFORollover(void);
void MAX30102_SetLEDMode(uint8_t mode);
void MAX30102_SetADCRange(uint8_t adcRange);
void MAX30102_SetSampleRate(uint8_t sampleRate);
void MAX30102_SetPulseWidth(uint8_t pulseWidth);
void MAX30102_SetPulseAmplitudeRed(uint8_t amplitude);
void MAX30102_SetPulseAmplitudeIR(uint8_t amplitude);
void MAX30102_SetPulseAmplitudeGreen(uint8_t amplitude);
void MAX30102_SetPulseAmplitudeProximity(uint8_t amplitude);
void MAX30102_EnableSlot(uint8_t slotNumber, uint8_t device);
void MAX30102_ClearFIFO(void);
void MAX30102_Setup(uint8_t  powerLevel, uint8_t sampleAverage, uint8_t ledMode,
void MAX30102_SoftReset(void);
bool MAX30102_ReadChannels(uint16_t *channelsBuffer, bool readFIFO);
bool MAX30102_SafeCheck(uint8_t maxTimeToCheck);
```

- Import the source file (MAX30102.c) in CM4 (Core1) → Source Files
- Repeat the process for the header file (MAX30102.h). Import the file in CM4 (Core1) → Header Files

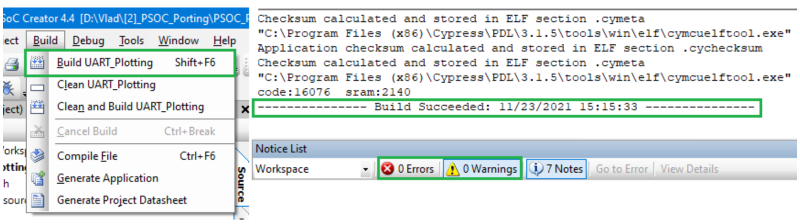
OLED library

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



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

Heart rate library

The library is provided by SparkFun for the MAX3010X sensor module and can extract the heart rate from PPG signals. The library is a modified version of the original one such that the HR value to be computed using the custom analog module from PPG EduKit platform.

- Import the source file (**heartRate.c**) in CM4 (Core1) → Source Files
- Repeat the process for the header file (**heartRate.h**) file. Import the files in CM4 (Core1) → Header Files
- Build the project to check if there is any error.
- Take a look at the application programming interface (**heartRate.c**) and try to understand the functions.

The heart rate algorithm is described below. Each sample is processed by an average DC estimator that acts as an approximate running average for every new value that is measured, then the DC average is subtracted from the current sample. Doing this, the signal now can also take negative values and the zero crossing points can be detected. For every cycle the maximum and minimum value is computed. A beat is detected on each rising edge when the previous cycle had a difference between the maximum and minimum value between 10 and 120. These values can be tweaked manually or an algorithm can be implemented to change them at runtime. The HR can be computed by measuring the time distance between two consecutive time beats detected by this function.

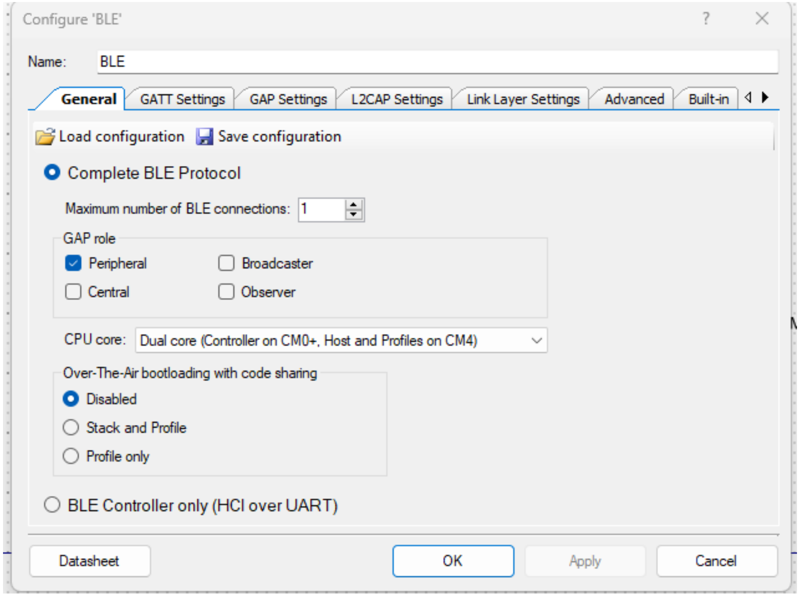
```
bool CUSTOM_checkForBeat(uint32_t sample)
{
  bool beatDetected = false;

  //  Save current state
  IR_AC_Signal_Previous = IR_AC_Signal_Current;

  //  Process next data sample
  IR_Average_Estimated = averageDCEstimator(&ir_avg_reg, sample);
  IR_AC_Signal_Current = sample - IR_Average_Estimated;

  //  Detect positive zero crossing (rising edge)
  if ((IR_AC_Signal_Previous < 0) & (IR_AC_Signal_Current >= 0))
  {

    IR_AC_Max = IR_AC_Signal_max; //Adjust our AC max and min
    IR_AC_Min = IR_AC_Signal_min;

    positiveEdge = 1;
    negativeEdge = 0;
    IR_AC_Signal_max = 0;

    if (((IR_AC_Max - IR_AC_Min) > 10) & ((IR_AC_Max - IR_AC_Min) < 120))
    {
      //Heart beat!!!
      beatDetected = true;
    }
  }

  //  Detect negative zero crossing (falling edge)
  if ((IR_AC_Signal_Previous > 0) & (IR_AC_Signal_Current <= 0))
  {
    positiveEdge = 0;
    negativeEdge = 1;
    IR_AC_Signal_min = 0;
  }

  //  Find Maximum value in positive cycle
  if (positiveEdge & (IR_AC_Signal_Current > IR_AC_Signal_Previous))
  {
    IR_AC_Signal_max = IR_AC_Signal_Current;
  }

  //  Find Minimum value in negative cycle
  if (negativeEdge & (IR_AC_Signal_Current < IR_AC_Signal_Previous))
  {
    IR_AC_Signal_min = IR_AC_Signal_Current;
  }

  return(beatDetected);
}
```

BLE integration

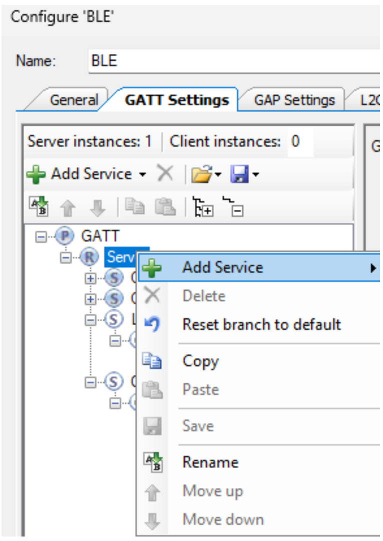
The Bluetooth Low Energy (BLE) Peripheral Driver Library (PDL) Component provides a comprehensive GUI-based configuration window to facilitate designing applications requiring BLE connectivity. BLE is used in very low-power network and Internet of Things (IoT) solutions aimed for low-cost battery operated devices that can quickly connect and form simple wireless links.

Bluetooth Low Energy devices transfer data back and forth using concepts called Services and Characteristics. It makes use of a generic data protocol called the Attribute Protocol (ATT), which is used to store Services, Characteristics and related data in a simple lookup table using 16-bit IDs for each entry in the table. PSOC Creator supports numerous SIG-adopted GATT-based Profiles and Services. Each of these can be configured for either a GATT Client or GATT Server. The Component generates all the necessary code for a particular Profile/Service operation, as configured in the Component Configure dialog.

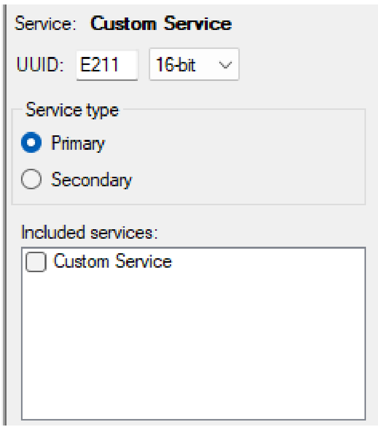
- Search for **BLE** and drag and drop the component into the schematic.
- Configure the general settings of the BLE module.



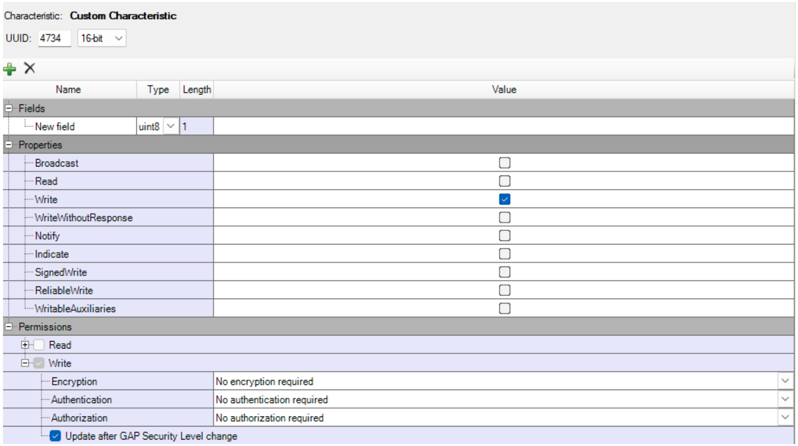
- Configure the Attribute Protocol. The board will act as a GATT Server which holds the ATT lookup data and service and characteristic definitions, and the GATT Client (the phone/tablet), which sends requests to this server. Right click on the GATT Server option and add a custom service (**Add Service** → **Custom Service**).



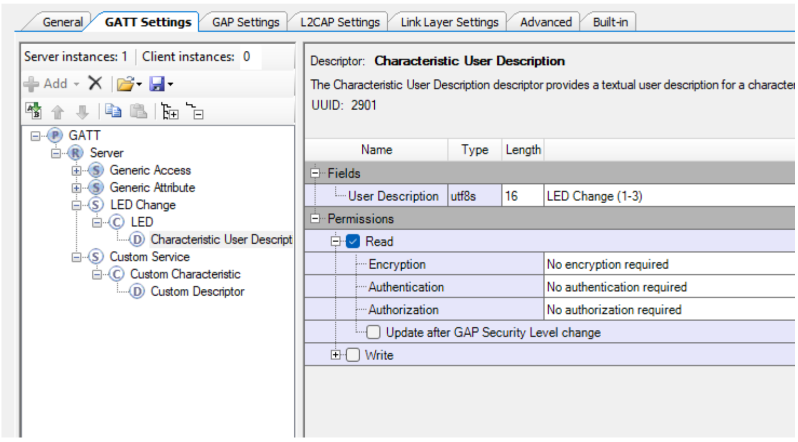
- Set the UUID of the service as follows.



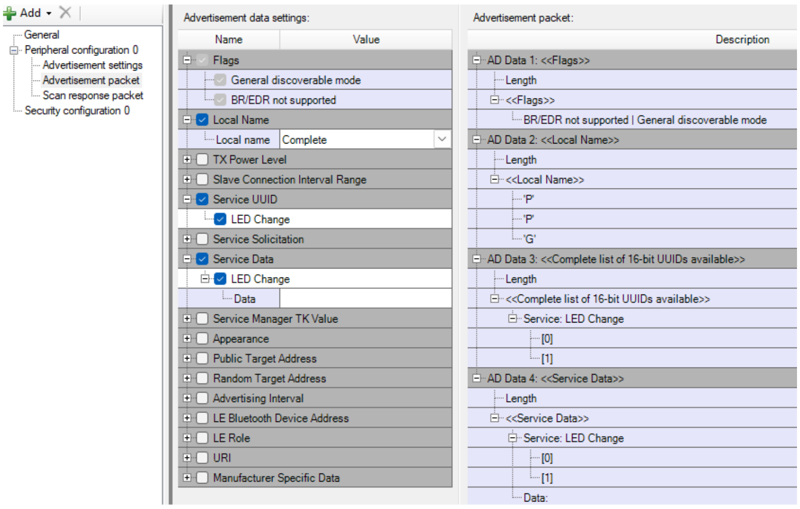
- Set the Custom Characteristic of the service as follows.



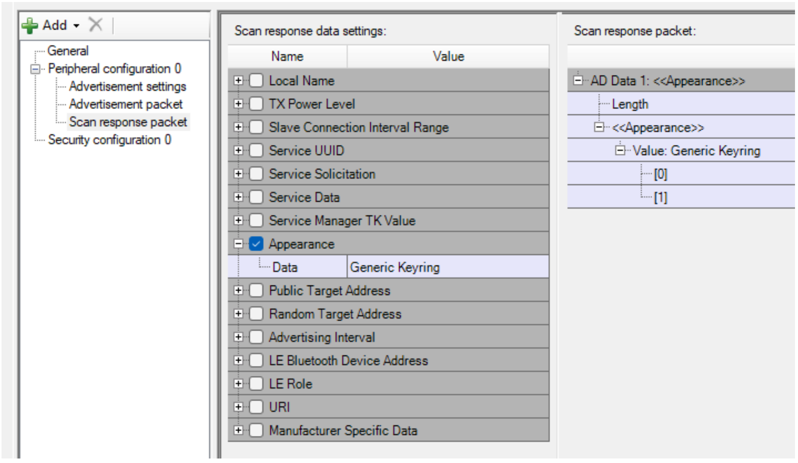
- Set the Characteristic User Description of the characteristic as follows.



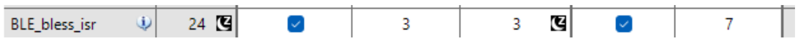
- Go to GAP Settings → General and a name for the device.
- Set the advertisements data settings of the BLE component.



- Set the scanning data settings of the BLE component.



- Enable the interrupt for both cores in the Interrupts tab. (CM0 is the controller, CM4 holds the host and profiles of the BLE).



3 Create the main program

The main program should merge together all the libraries in such a way to reach the goal of measuring the data from both sensors and to compute the HR value. The values will be displayed on the OLED display for both sensors and the wavelength of the measurement can be controlled over the BLE. The main function can be found in **main_c4.c**.

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

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

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

/* @brief Include PSOC generated files */
#include "project.h"

/* @brief Include custom libraries for PPG EduKit */
#include "utils.h"
#include "MAX30105.h"
#include "milliseconds.h"
#include "heartRate.h"
#include "custom_ppg.h"
#include "AD5273.h"
#include "TLC5925.h"
#include "custom_ppg.h"
#include "oled_driver.h"
#include "serial_frame.h"
```

- Declare local macros. These macro directives are used for MAX30102 configuration and heart rate buffer size.

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

/* @brief Activate MAX30105 sensor. If FALSE, only the custom sensor will be used */
#define USE_MAX_PPG_SENSOR        TRUE

/* @brief Enable UART interface */
#define SERIAL_DEBUG              TRUE

/* @brief Size of averaging buffer for HR and SpO2 measurements */
#define BPM_AVERAGE_RATE          (6U)
#define SPO2_AVERAGE_RATE         (3U)

/* @brief MAX30105 Options: 0=Off to 255=50mA */
#define CFG_LED_BRIGHTNESS    (0x1f)
/* @brief MAX30105 Options: 1, 2, 4, 8, 16, */
#define CFG_SAMPLE_AVERAGE    (4)
/* @brief MAX30105 Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green */
#define CFG_LED_MODE          (2)
/* @brief MAX30105 Options: 50, 100, 200, 400, 800, 1000, 1600, 3200 */
#define CFG_SAMPLE_RATE       (400)
/* @brief MAX30105 Options: 69, 118, 215, 411 */
#define CFG_PULSE_WIDTH       (411)
/* @brief MAX30105 Options: 2048, 4096, 8192, 16384 */
#define CFG_ADC_RANGE         (4096)

/* @brief Cursor (X,Y) position for OLED custom text */
#define OLED_MAX30105_TEXT_X_POSITION       (0U)
#define OLED_MAX30105_TEXT_Y_POSITION       (10U)
#define OLED_CUSTOM_TEXT_X_POSITION         (0U)
#define OLED_CUSTOM_TEXT_Y_POSITION         (30U)
#define OLED_HR_VALUE_TEXT_X_POSITION       (60U)
```

- Declare local functions used to print the text on the display. Declare also a BLE callback function that will be called for every triggered BLE event.

```c
/*==========================================================================================
 *                                   LOCAL FUNCTION PROTOTYPES
 *   ==========================================================================================*/

static void clearHrDigits(void);
static void displayHRText(void);
static void refreshHRValues(void);

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

void BLE_callback(uint32_t event, void *eventParam);

```

- The HR values will be printed on the display continuously. Before printing any new value the display region that contains the old value should be erased (set to black). This is achieved using **clearHRDigits** function that is described below. The **displayHRText** function only prints a text message during the initialization and it is called only once.

```c
/**************************************************************************
 * Function: clearHrDigits
 *
 * Description: Prepare OLED region for new HR values. Delete old values.
 **************************************************************************/
static void clearHrDigits(void)
{
    for(uint8_t i = 70; i < 90; i++)
    {
        for(uint8_t j = OLED_MAX30105_TEXT_Y_POSITION; j < (OLED_MAX30105_TEXT_Y_POSITION + 20); j++)
            gfx_drawPixel(i, j, BLACK);
        for(uint8_t j = OLED_CUSTOM_TEXT_Y_POSITION; j < (OLED_CUSTOM_TEXT_Y_POSITION + 20) ; j++)
            gfx_drawPixel(i, j, BLACK);
    }
    display_update();
}

/**************************************************************************
 * Function: displayHRText
 *
 * Description: Prepare OLED region for new HR values. Delete old values.
 **************************************************************************/
static void displayHRText(void)
{
    gfx_setTextSize(1);
    gfx_setTextColor(WHITE);
    gfx_setCursor(OLED_MAX30105_TEXT_X_POSITION, OLED_MAX30105_TEXT_Y_POSITION);
    for(uint8_t i = 0 ; i < sizeof(oledText[0]) ; i++){
            gfx_write(oledText[0][i]);
      }
    gfx_setCursor(OLED_CUSTOM_TEXT_X_POSITION, OLED_CUSTOM_TEXT_Y_POSITION);
    for(uint8_t i = 0 ; i < sizeof(oledText[1]) ; i++){
            gfx_write(oledText[1][i]);
    }
    display_update();
}
```

- After the display region that contains the HR values is erased new values can be written using **refreshHRValues** function.

```c
/*************************************************************************
 * Function: refreshHRValues
 *
 * Description: Display new HR values on the OLED display
 ************************************************************************/
static void refreshHRValues(void)
{
    /* Clear old HR values */
    clearHrDigits();

    /* Prepare to display new MAX30105 HR value */
    gfx_setCursor(OLED_HR_VALUE_TEXT_X_POSITION, OLED_MAX30105_TEXT_Y_POSITION);
    /* Convert dec number to ASCII value */
    display_usint2decascii(gTempbeatAvg, numdec_buffer);
    /* Draw the ASCII HR value and store it in _displaybuf */
    for(uint8_t i = 0 ; i < sizeof(numdec_buffer) ; i++){
        gfx_write(numdec_buffer[i]);
    }

    /* Prepare to display new custom HR value */
    gfx_setCursor(OLED_HR_VALUE_TEXT_X_POSITION, OLED_CUSTOM_TEXT_Y_POSITION);
    /* Convert dec number to ASCII value */
    display_usint2decascii(gCustomTempbeatAvg, numdec_buffer);
    /* Draw the ASCII HR value and store it in _displaybuf */
    for(uint8_t i = 0 ; i < sizeof(numdec_buffer) ; i++){
        gfx_write(numdec_buffer[i]);
    }

    display_update();
}
```

- The main function starts with the initialization of **adcRead_isr** and **isr_1ms** interrupts by setting the priority and the aferent interrupt vector. Then, the interrupts are enabled and the UART peripheral is initialized. Besides this, many other local variables are declared and used later during HR computation.

```
/*==============================================================================================
 *                                         GLOBAL FUNCTIONS
 *   ============================================================================================*/

int main(void)
{
    const uint8_t RATE_SIZE = 6; //Increase this for more averaging.

    uint8_t rates1[RATE_SIZE]; //Array of heart rates
    uint8_t rateSpot1 = 0;
    long lastBeat1 = 0; //Time at which the last beat occurred
    float beatsPerMinute1;
    int beatAvg1;

    uint8_t rates2[RATE_SIZE]; //Array of heart rates
    uint8_t rateSpot2 = 0;
    long lastBeat2 = 0; //Time at which the last beat occurred
    float beatsPerMinute2;
    int beatAvg2;

    /* Assign ISR routines */
    MILLIS_AssignISR();
    CUSTOM_PPG_AssignISR_Spo2();
    CUSTOM_PPG_AssignISR_AdcRead();

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

    /* Start timer used for delay function */
    MILLIS_InitAndStartTimer();

#if (SERIAL_DEBUG == TRUE)
    /* UART initialization status */
    cy_en_scb_uart_status_t uart_status ;
     /* Initialize UART operation. Config and Context structure is copied from Generated source. */
    uart_status  = Cy_SCB_UART_Init(Uart_Printf_HW, &Uart_Printf_config, &Uart_Printf_context);
    if(uart_status != CY_SCB_UART_SUCCESS)
    {
        HandleError();
    }
    Cy_SCB_UART_Enable(Uart_Printf_HW);
#endif
```

- Further, the PPG EduKit shield is initialized (digital potentiometer, LED driver, ADC) and the BLE is started having as callback the function declared previously.

```
    /* Init digital potentiometer */
    AD5273_Init();
    /* Set 5 mA current for the LED driver */
    TLC5925_SetCurrent_mA(20);
    /* Enable IR LED */
    TLC5925_enableGreen();

    /* Start ADC and ADC Conversion */
    ADC_Start();

    /* Start timer used for ADC reading */
    CUSTOM_PPG_InitAndStartTimer_AdcRead();

    /* Wait for VCC stable before initializing the OLED display */
    CyDelay(1000);
    /* Init 128x64 OLED FeatherWing display. */
    display_init();

    CyDelay(1000);
    display_clear();
    display_update();

    /* Set display rotation to 1 (width -> height, height -> width) */
    gfx_setRotation(1);
    CyDelay(50);
    /* Display custom messages */
    displayHRText();

    numdec_buffer[USINT2DECASCII_MAX_DIGITS] = '\0';

    Cy_BLE_Start(BLE_callback);

    while(Cy_BLE_GetState() != CY_BLE_STATE_ON)
    {
        Cy_BLE_ProcessEvents();
    }

#if (USE_MAX_PPG_SENSOR == TRUE)
    /* Initialize MAX30105 sensor */
    MAX30105_Setup(CFG_LED_BRIGHTNESS, CFG_SAMPLE_AVERAGE, CFG_LED_MODE, CFG_SAMPLE_RATE, CFG_PULSE_WIDTH, CFG_ADC_RANGE);
#endif
```

- Firstly, the ADC value is read from the output of the inverting amplifier and each sample is used to update the beat searching algorithm. If a beat is detected, the algorithm computes the BPM value and keeps a moving average of the values.

```
    while(1)
    {
        CyDelay(100);
        long ppgValue1 =  ADC_GetResult16(ADC_CHANNEL_0_INV_AMP);
        //printf("%ld\n", ppgValue1);
        if (CUSTOM_checkForBeat(ppgValue1) == true)
        {

          long delta1 = MILLIS_GetValue() - lastBeat1;
          lastBeat1 = MILLIS_GetValue();

          beatsPerMinute1 = 60 / (delta1 / 1000.0);

          if (beatsPerMinute1 < 255 && beatsPerMinute1 > 20)
          {
            rates1[rateSpot1++] = (uint8_t)beatsPerMinute1; //Store this reading in the array
            rateSpot1 %= RATE_SIZE; //Wrap variable

            //Take average of readings
            beatAvg1 = 0;
            for (uint8_t x = 0 ; x < RATE_SIZE ; x++)
              beatAvg1 += rates1[x];
            beatAvg1 /= RATE_SIZE;
            gCustomTempbeatAvg = beatAvg1;
          }

        refreshHRValues();
     }

```

- The same principle is applied for MAX30105 sensor, but this time the function that is used to read the value from the I2C connected sensor is different.

```c
    long ppgValue2 =  MAX30105_GetRed();
    //printf("%ld\n", ppgValue2);

    if(ppgValue2 > 50000)
    {

        if (checkForBeat(ppgValue2) == true)
        {

            long delta2 = MILLIS_GetValue() - lastBeat2;
            lastBeat2 = MILLIS_GetValue();

            beatsPerMinute2 = 60 / (delta2 / 1000.0);

            if (beatsPerMinute2 < 255 && beatsPerMinute2 > 20)
            {
                rates2[rateSpot2++] = (uint8_t)beatsPerMinute2; //Store this reading in the array
                rateSpot2 %= RATE_SIZE; //Wrap variable

                //Take average of readings
                beatAvg2 = 0;
                for (uint8_t x = 0 ; x < RATE_SIZE ; x++)
                  beatAvg2 += rates2[x];
                beatAvg2 /= RATE_SIZE;
                gTempbeatAvg = beatAvg2;
            }

            refreshHRValues();
        }
    }
   }
```

- In the end, to control the wavelength over BLE, the callback function should be declared. The function takes as parameters an event variable and a void pointer that keeps the address of the parameters received over BLE. The most important event here would be the write request that comes from the client application. There could be many attributes written during a write request, but here we added only one attribute (LED_CHANGE). Depending on the value written by the client application, the current and the LED wavelength is set accordingly.

```c
void BLE_callback(uint32_t event, void *eventParam)
{
    cy_stc_ble_gatts_write_cmd_req_param_t  *writeReqParameter;
switch (event)
{
    case CY_BLE_EVT_STACK_ON:
        printf("Start advertising\n");
        Cy_BLE_GAPP_StartAdvertisement(CY_BLE_ADVERTISING_FAST, CY_BLE_PERIPHERAL_CONFIGURATION_0_INDEX);
    break;

    case CY_BLE_EVT_GAP_DEVICE_DISCONNECTED:
        printf("Device disconnected!\n");
        Cy_BLE_GAPP_StartAdvertisement(CY_BLE_ADVERTISING_FAST, CY_BLE_PERIPHERAL_CONFIGURATION_0_INDEX);
        printf(">>Start advertising!\n");
        break;

    case CY_BLE_EVT_GATT_CONNECT_IND:
        printf("Device connected!\n");
    break;

    case CY_BLE_EVT_GATTS_WRITE_REQ:

        writeReqParameter = (cy_stc_ble_gatts_write_cmd_req_param_t *)eventParam;

        if(CY_BLE_LED_CHANGE_LED_CHAR_HANDLE == writeReqParameter->handleValPair.attrHandle)
        {
            uint8_t val = writeReqParameter->handleValPair.value.val[0];

            if(val==49)
            {
                TLC5925_enableIR();
                TLC5925_SetCurrent_mA(5);
            }
            else if(val==50)
            {
                TLC5925_enableRed();
                TLC5925_SetCurrent_mA(10);
            }
            else if (val == 51) {
                TLC5925_enableGreen();
                TLC5925_SetCurrent_mA(20);
            }
            else TLC5925_enableGreen();
        }
        Cy_BLE_GATTS_WriteRsp(writeReqParameter->connHandle);
        break;
    default:
        TLC5925_enableGreen();
        break;
```

- Download any mobile application that is intended to control devices over BLE. Power on the PPG EduKit, search for your device and check for the existence of the custom service already created. Search for the write attribute option and the number of the wavelength that you want to use (49, 50, 51). Depending on the mobile application, the values might need to be converted to HEX before sending the request.

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