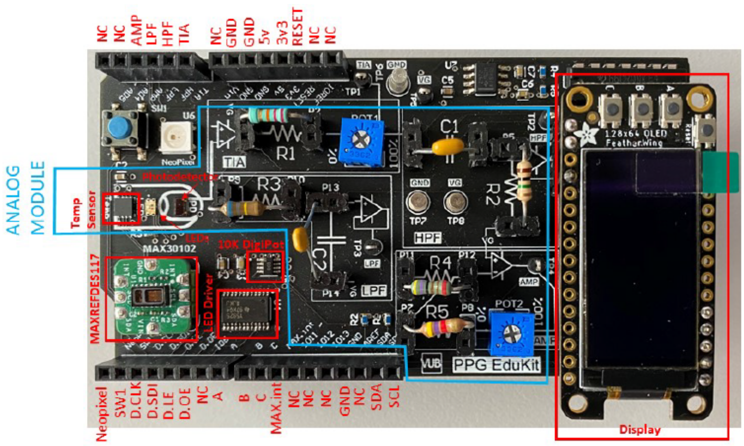
Lab 5: SpO2 algorithms and implementation

Introduction

This lab helps the user to understand the basics of SpO2 measurement and how to implement different algorithms to determine the blood oxygen saturation. Besides that, a comparison between the MAX30102 and the custom analog module is presented.

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.



SpO2 measurement

SpO2 stands for peripheral capillary oxygen saturation, an estimate of the amount of oxygen in the blood. More specifically, it is the percentage of oxygenated hemoglobin (hemoglobin containing oxygen) compared to the total amount of hemoglobin in the blood (oxygenated and non-oxygenated hemoglobin).

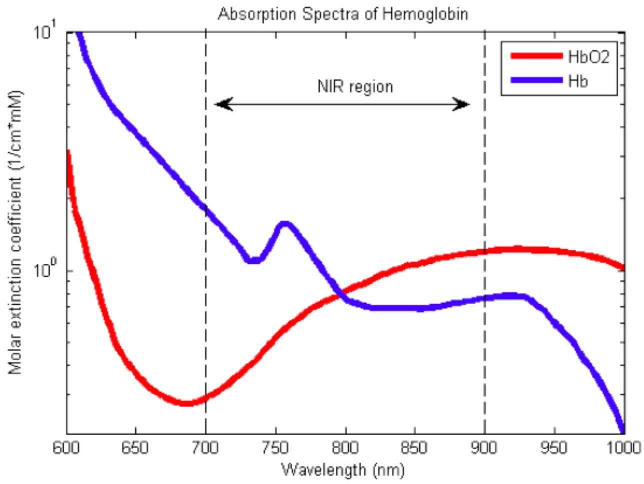
The measurement consists of exposing the tissue to a source of light that changes its wavelength at a frequency of hundred of hertz. The standard wavelengths that are used for SpO2 measurement are red (660nm) and IR (940nm).

Giving the fact that red and infrared wavelengths have a different absorption coefficient for oxyhemoglobin (HbO2) and deoxyhemoglobin (Hb), one can compute the ratio of absorptions and determine the oxygen saturation.

$$R = \frac{(AC/DC)_{red}}{(AC/DC)_{IR}} \tag{1}$$

Once the R value is calculated from the two PPG signals, SpO_2 values are determined from R by using equation 2.2 or by using a lookup table:

$$\%SpO_2 = 104 - 17 * R \tag{2}$$



By Adrian Curtin (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

Objectives

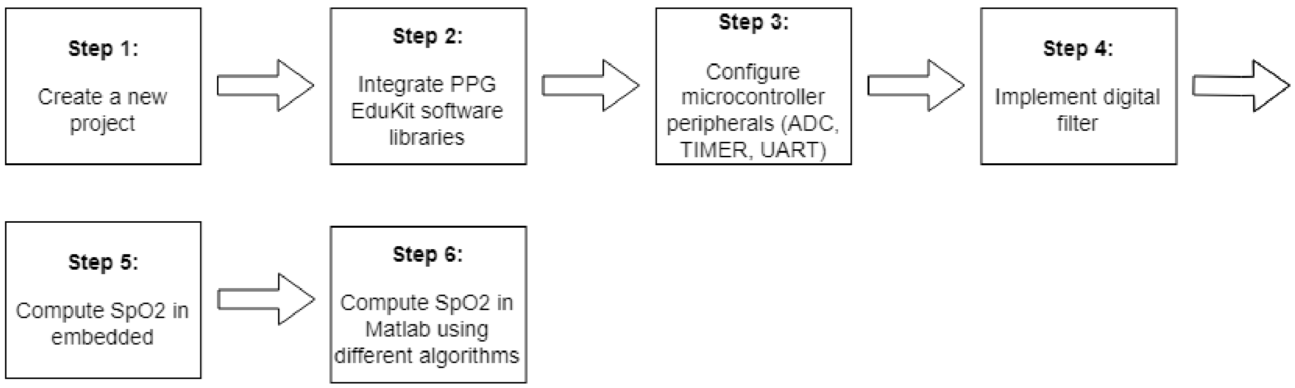
After completing this lab, you will be able to:

- Configure microcontroller peripherals in PSOC Creator (ADC, Timer, IRQ, UART)
- Understand multiwavelength measurement and control LED wavelength
- Configure and implement a digital filter
- Compute the SpO2 in embedded using one of the methods
- Compute the SpO2 in Matlab based on the acquired data

Procedure

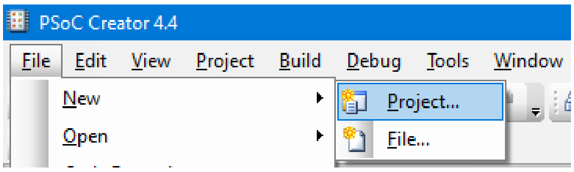
This lab is separated into steps that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

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 UART communication and interrupts, implement a digital filter, compute SpO2 and compare the sensor module with the analog module, and finally compute the SpO2 in Matlab using different methods.

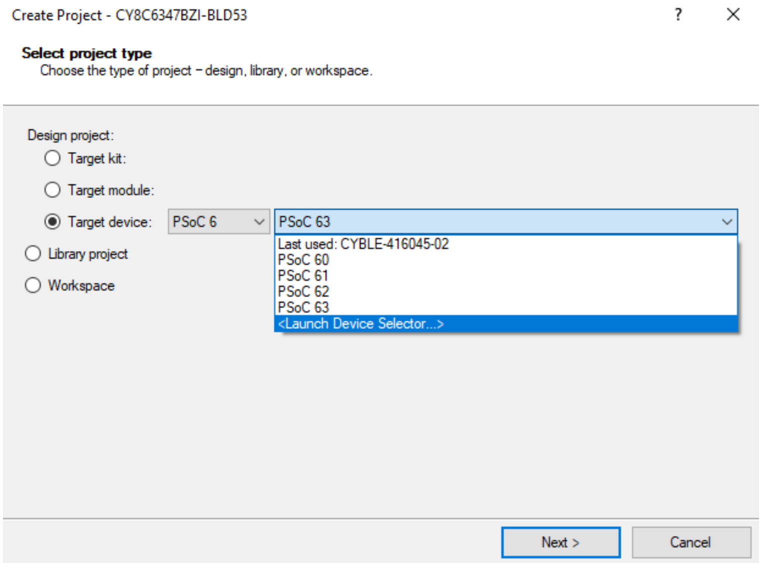


1 Creation of the Project

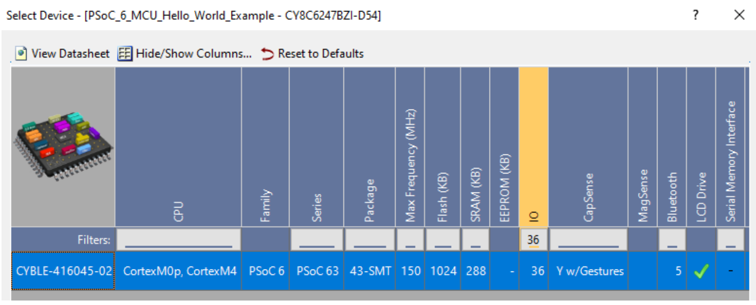
- Open PSoC Creator
- Go to File → New → Project



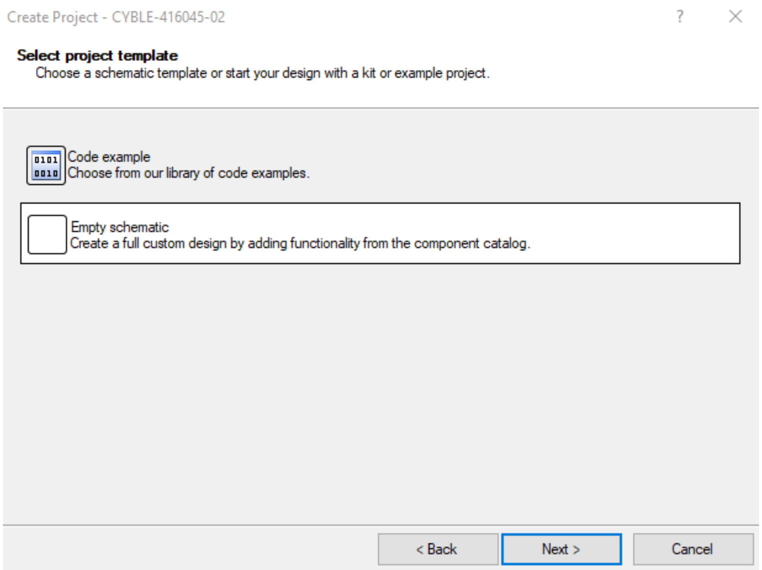
- Select the target device → PSOC6 → <Launch Device Selector>



- 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 the project name is **SpO2_measurement**

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 SpO2 using the PPG EduKit GUI the user has to integrate the following libraries: TLC5925 (LED driver library), AD5273 (digital potentiometer library), custom PPG, FIR filters, SpO2 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.

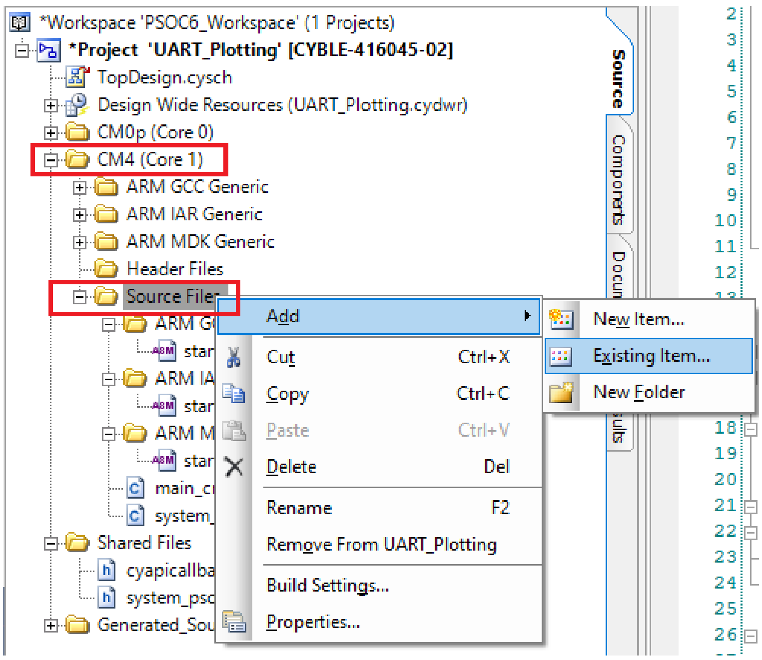
Utils library

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

```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

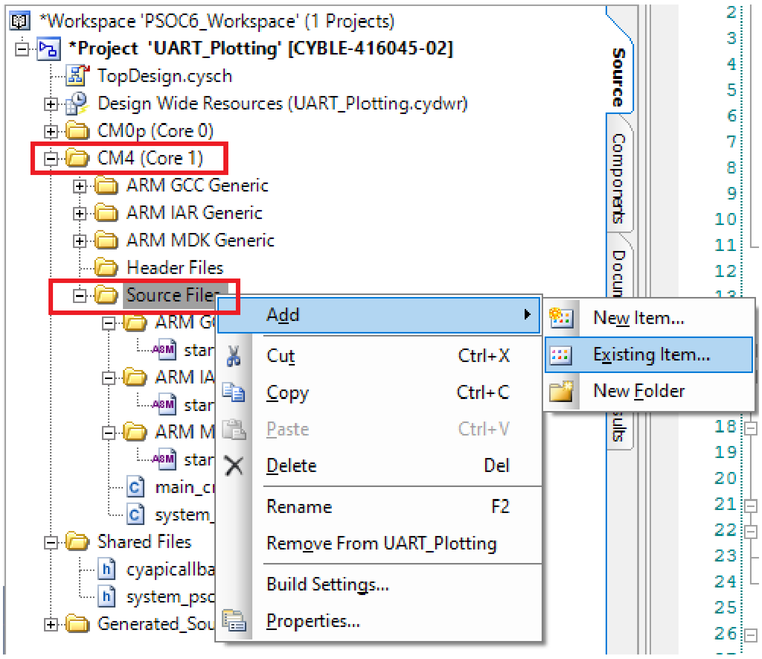


- Repeat the process for the header file (utils.h). Import the file in CM4 (Core1) → Header Files

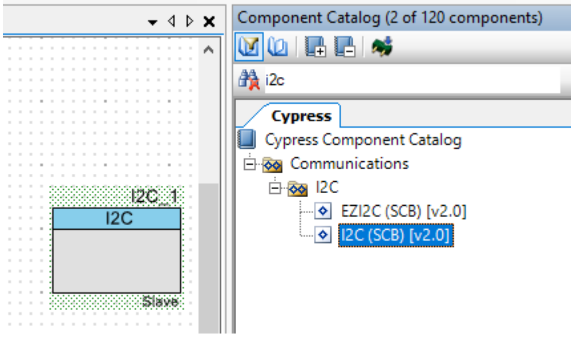
AD5273 driver

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.

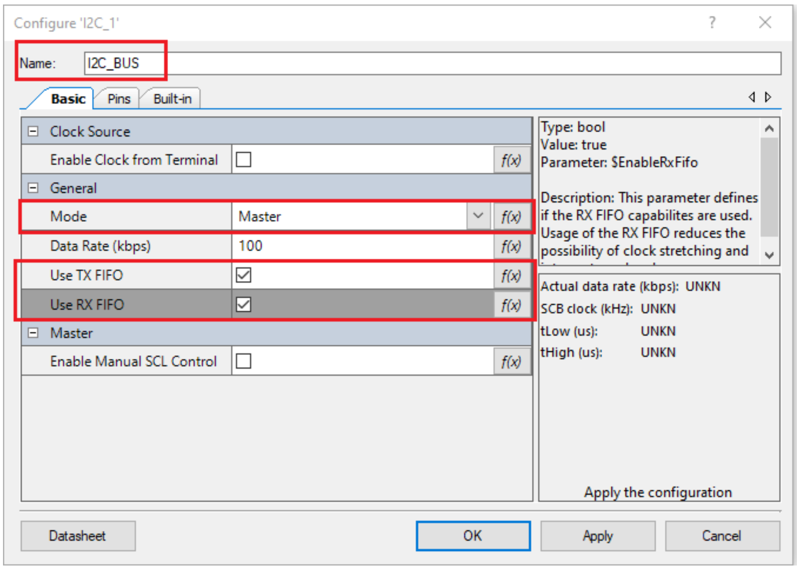
- Import the source file (AD5273.c) in CM4 (Core1) → Source Files



- 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.

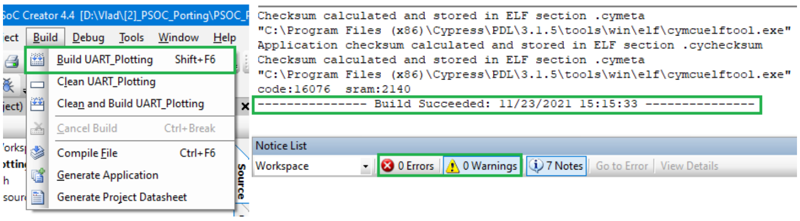


- Configure the I2C component by double click on it. Rename the component as **I2C_BUS**, set the mode as **master** and enable **TX/RX FIFO** buffers.



- 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.

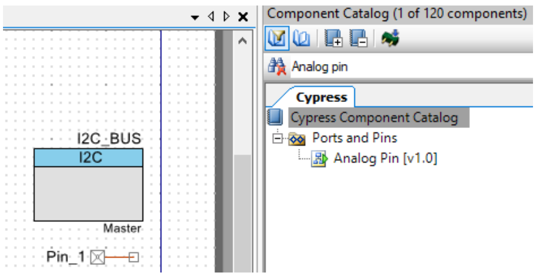


- Take a look at the application programming interface (AD5273.c) and try to understand driver functions.

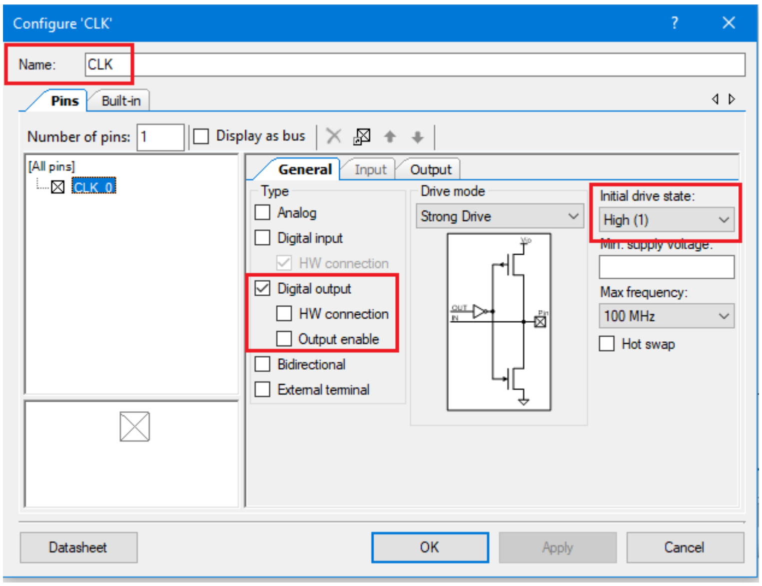
TLC5925 driver

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.

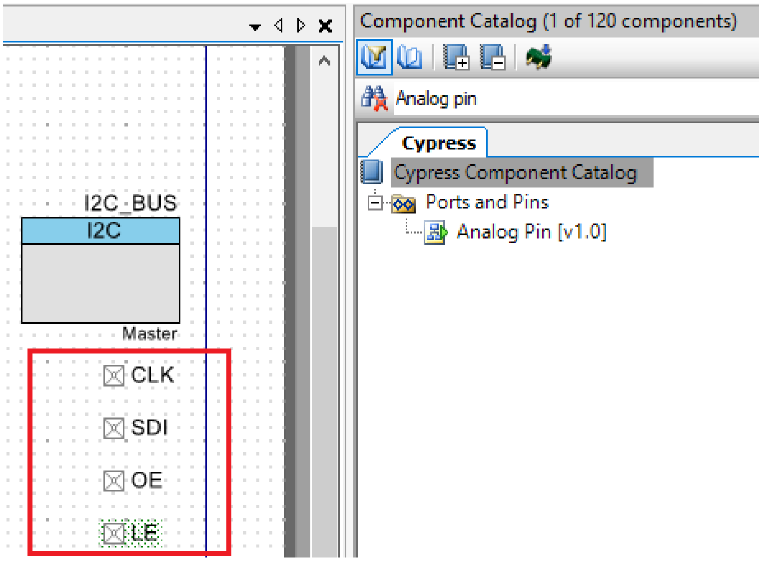
- Import the source file (TLC5925.c) in CM4 (Core1) → Source Files
- Repeat the process for the header file (TLC5925.h). Import the file in CM4 (Core1) → Header Files
- Go to the **TopDesign** schematic. In the component catalog (right panel at the right of the screen), write **Analog pin** and drag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



- Configure the Pin component by double click on it.



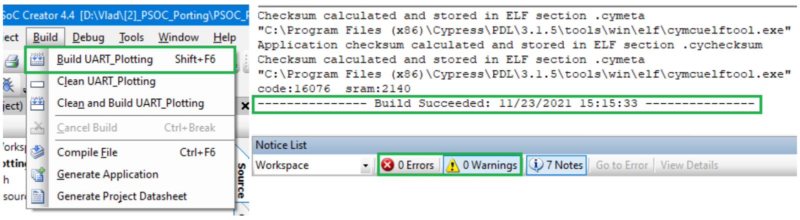
- Repeat the step for SDI, OE and LE pins.



- 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.

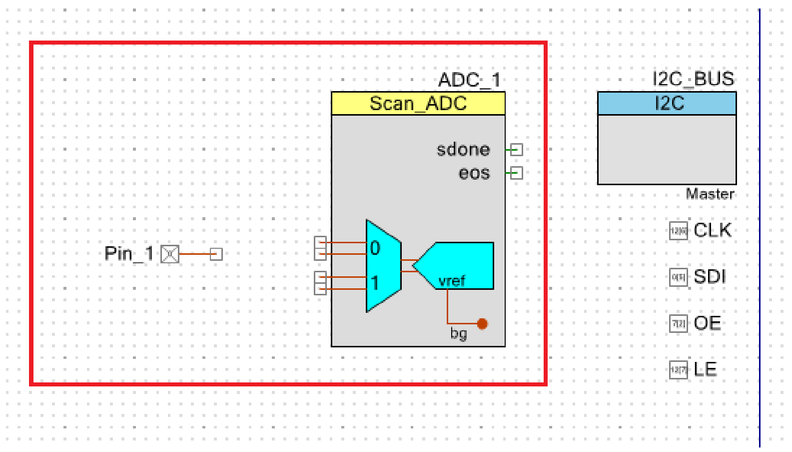


- Take a look at the application programming interface (TLC5925.c) and try to understand driver functions.

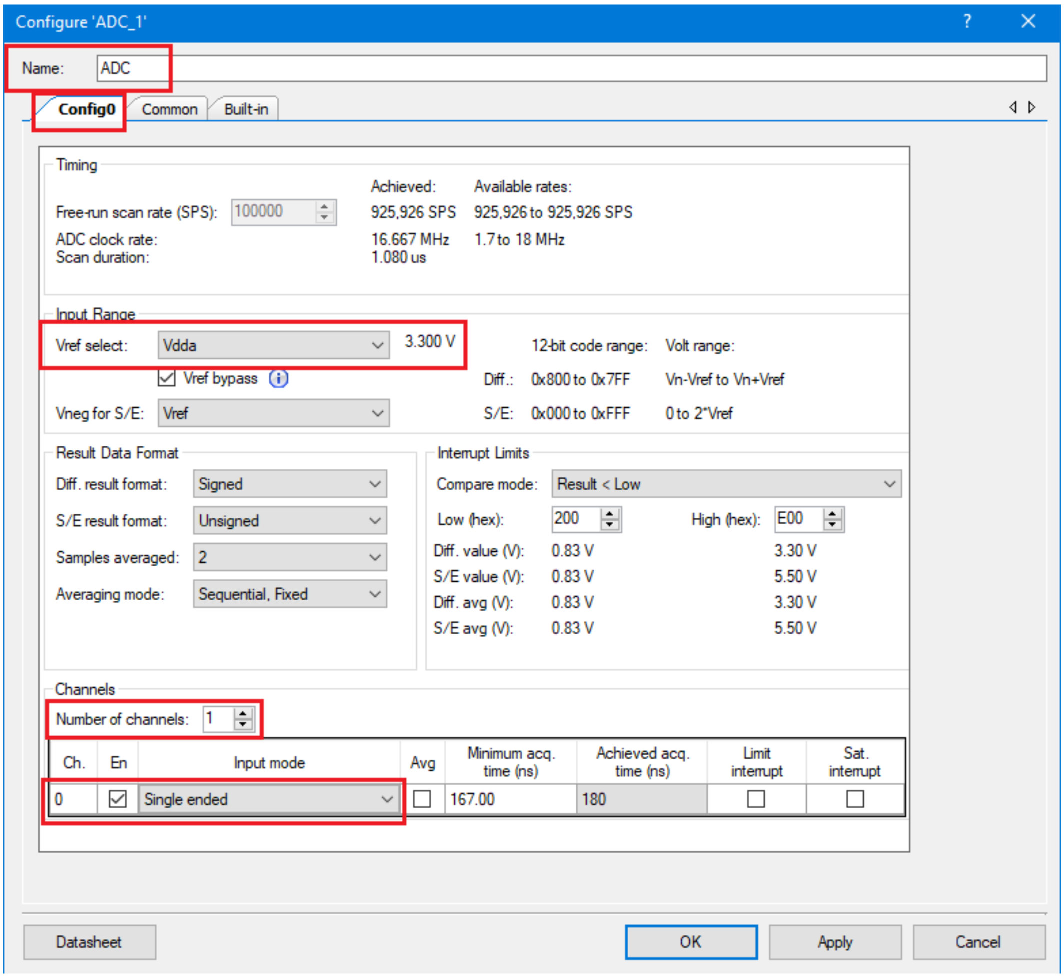
Custom PPG driver

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.

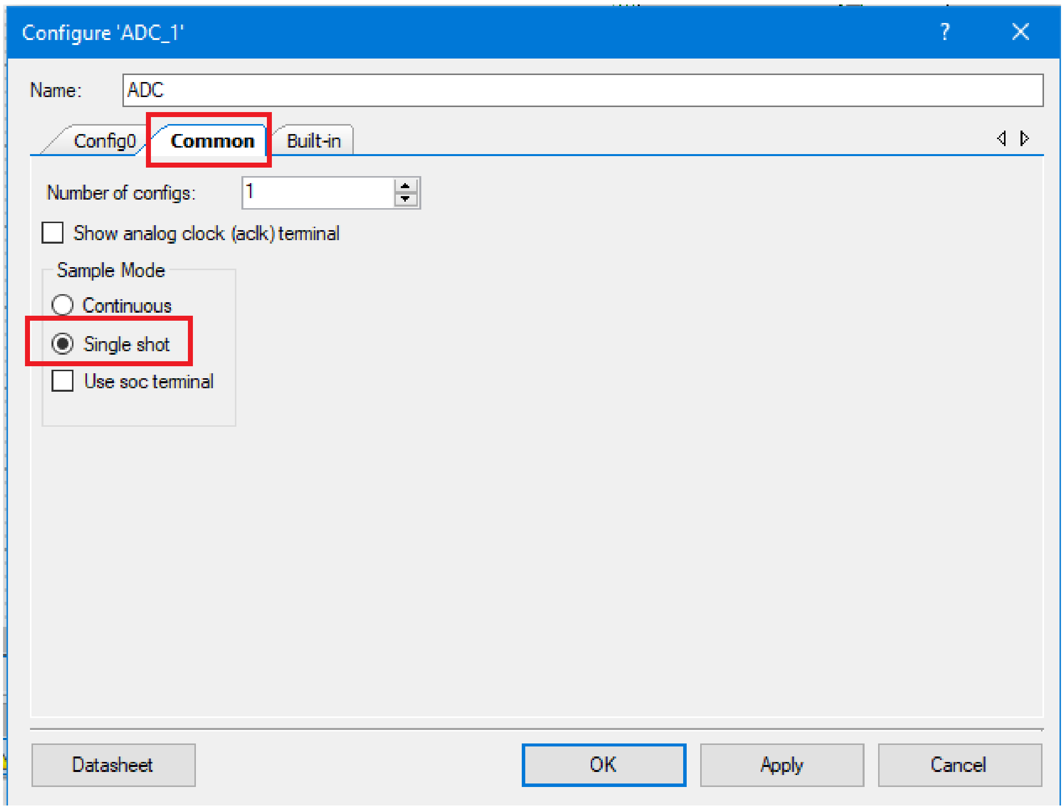
- Go to the **TopDesign** schematic. In the component catalog (right panel at the right of the screen), write **ADC** and drag and drop the component into the schematic. Search for **Analog pin** and drag and drop the component into the schematic.



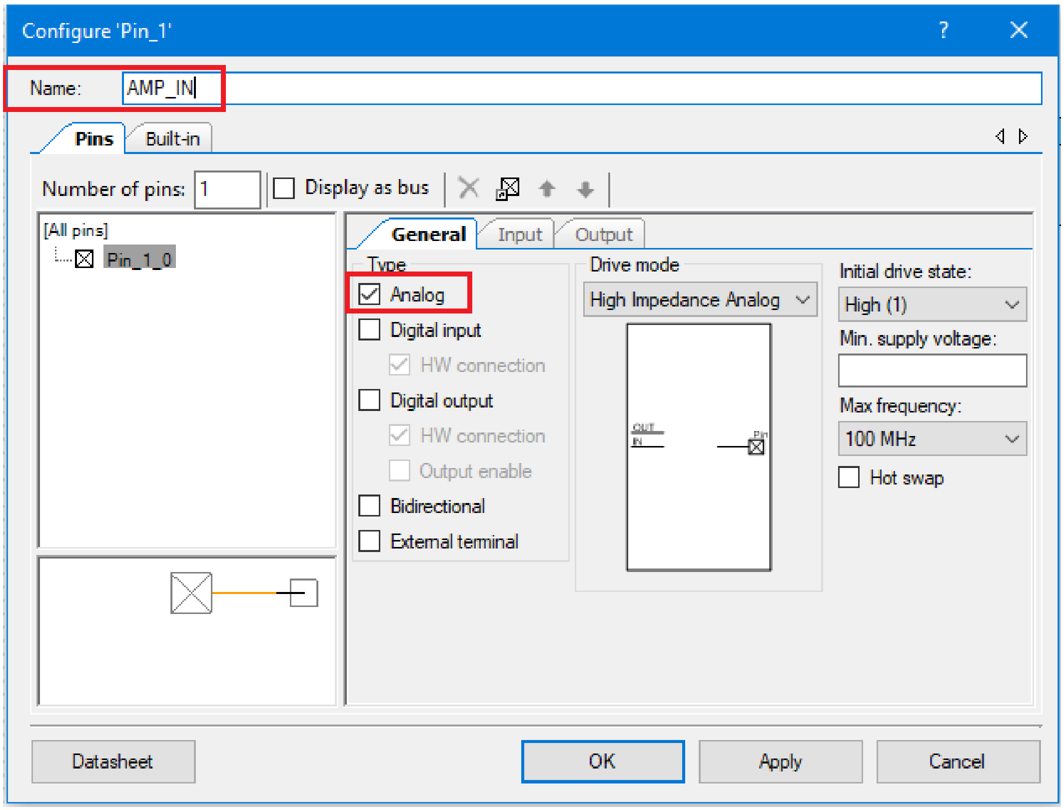
- Configure the ADC component by double click on it.



- Configure the ADC sample mode as single shot mode.



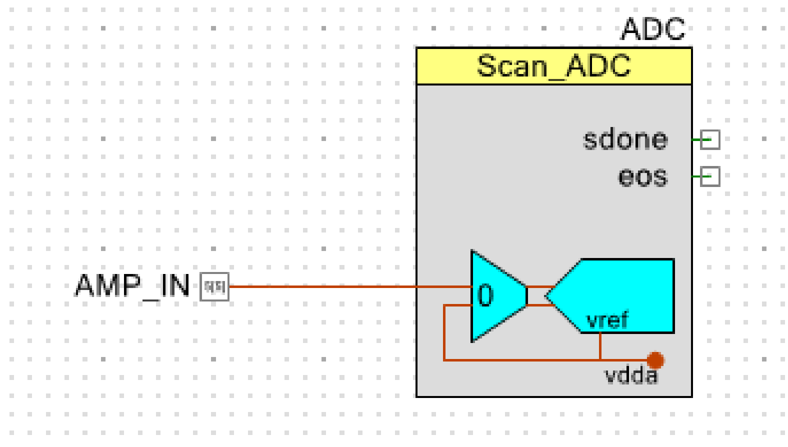
- Configure the Pin component by double click on it.



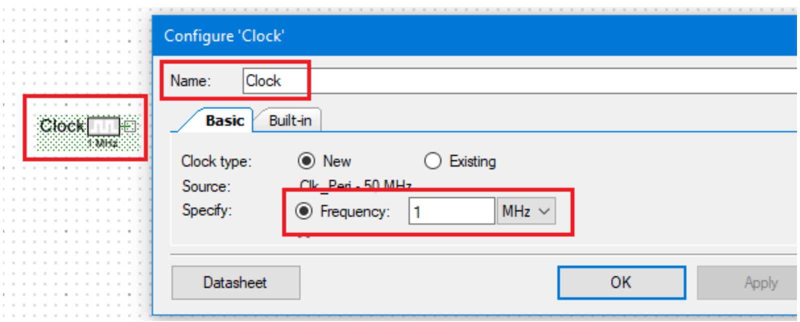
- Go to Design Wide Resources → Pins and assign the **AMP_IN** pin as follows:

AMP_IN	P5[2]	38	☑
CLK	P12[6]	34	☑
LE	P12[7]	35	☑
OE	P7[2]	22	☑
SDI	P0[5]	2	☑

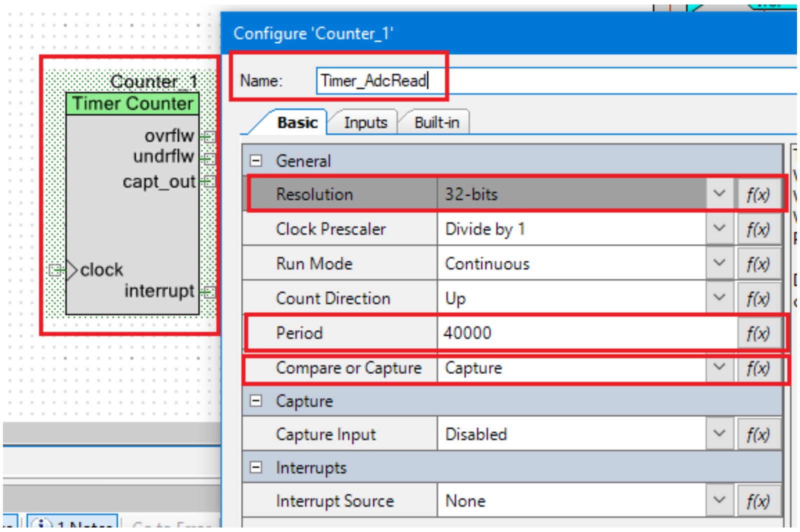
- Connect the analog pin to the ADC (press W to place a wire).



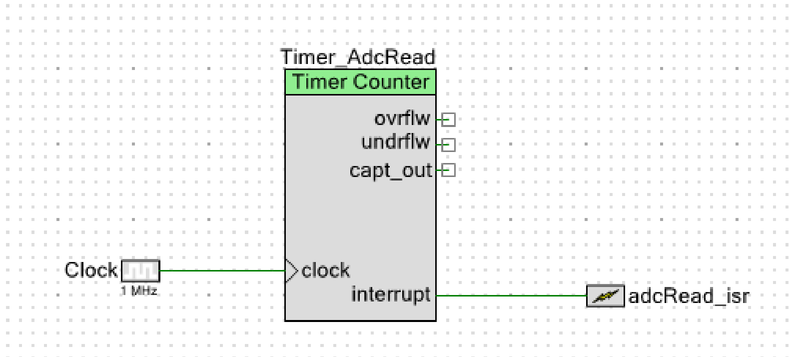
- Search for **Clock** and drag and drop the component into the schematic.
- Configure clock to 1 MHz



- 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

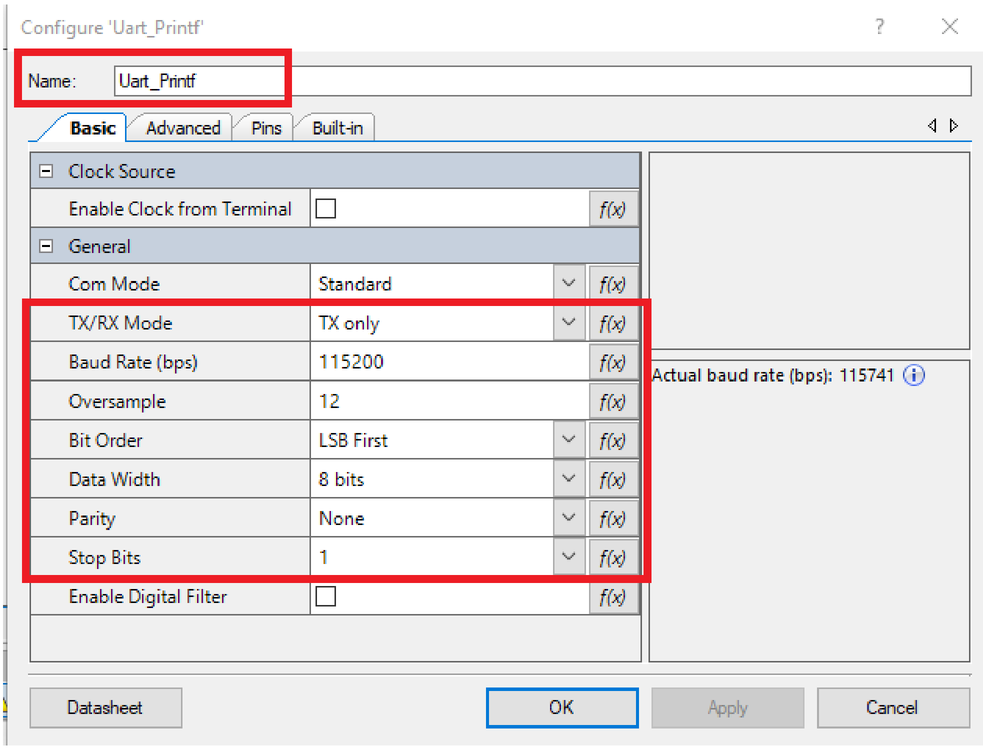


- Import the source file (custom_ppg.c) in CM4 (Core1) → Source Files
- Repeat the process for the header file (custom_ppg.h). Import the file in CM4 (Core1) → Header Files
- Build the project to check if there is any error.
- Take a look at the application programming interface (custom_ppg.c) and try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

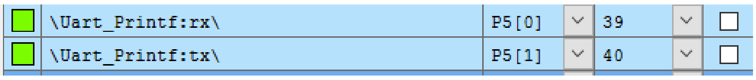
Serial Frame library

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:

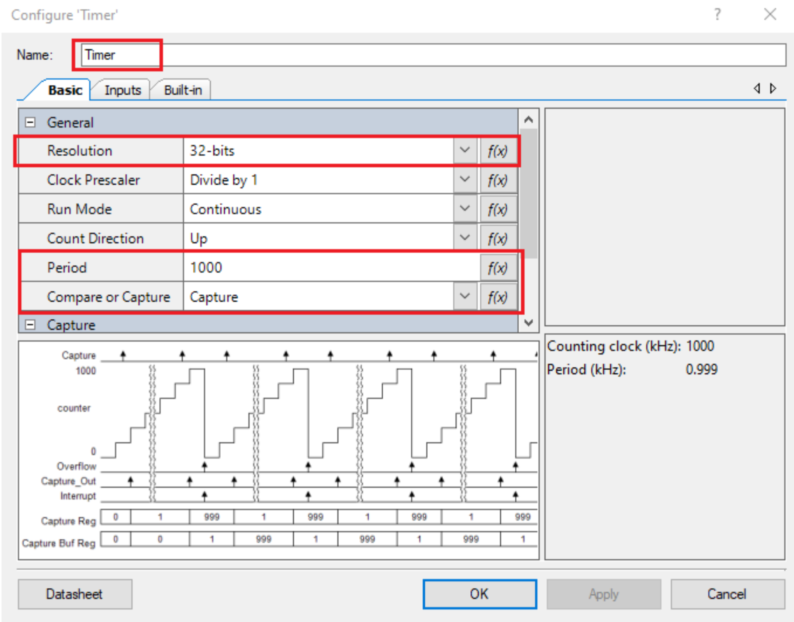


- Import the source file (serial_frame.c) in CM4 (Core1) → Source Files
- Repeat the process for the header file (serial_frame.h). Import the file in CM4 (Core1) → Header Files
- Build the project to check if there is any error.
- Take a look at the application programming interface (serial_frame.c)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

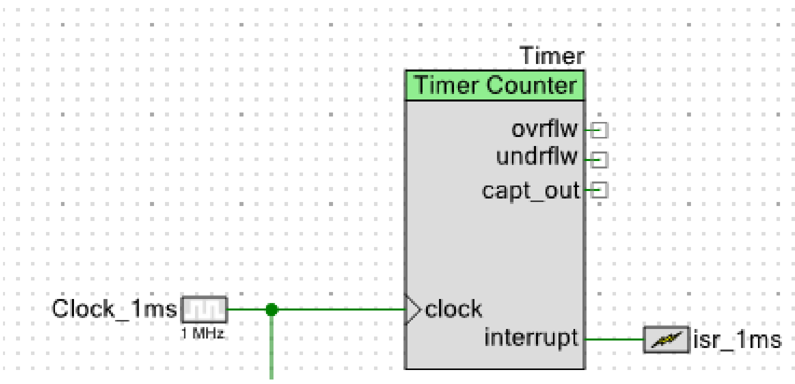
Milliseconds library

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 [4]. 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.

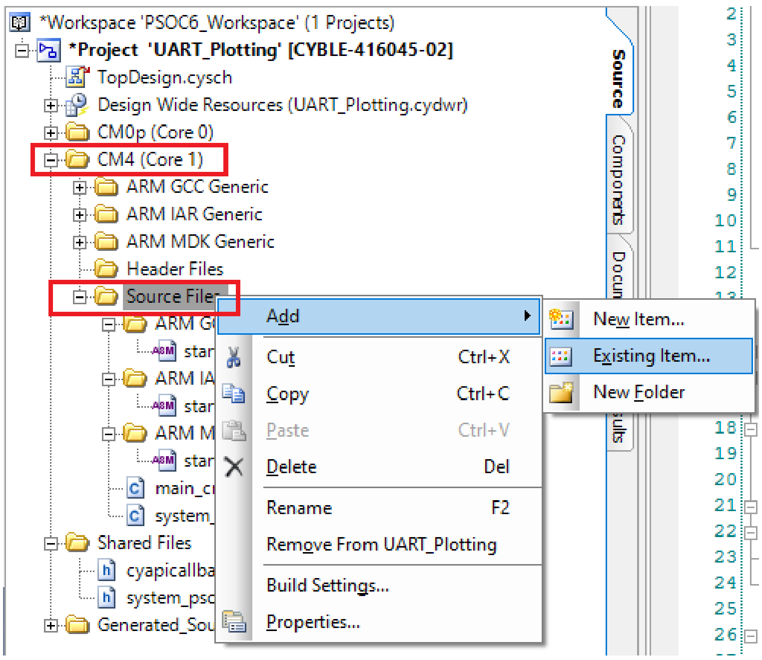
```
void MAX30102_NextSample(void);
void MAX30102_SetFIFOAverage(uint8_t numberOfSamples);
void MAX30102_EnableFIFORollover(void);
void MAX30102_SetLEDMode(uint8_t mode);
void MAX30102_SetADCRange(uint8_t adcRange);
void MAX30102_SetSampleRate(uint8_t sampleRate);
void MAX30102_SetPulseWidth(uint8_t 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

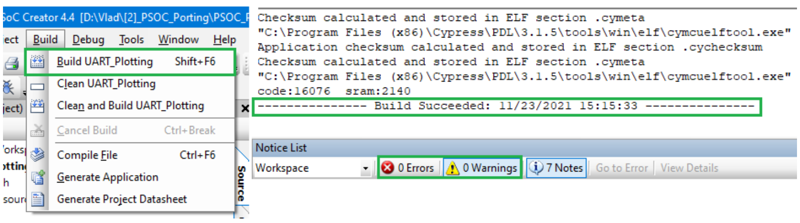
OLED library

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 driver functions.

SpO2 library

The library is provided by SparkFun for the MAX3010X sensor module and can extract the heart rate and SpO2 value from PPG signals. The library is a modified version of the original one such that the SpO2 value to be computed using the custom analog module from PPG EduKit platform.

- Import the source file (**spo2_algorithm.c**) in CM4 (Core1) → Source Files
- Repeat the process for the header file (**spo2_algorithm.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 (**spo2_algorithm.c**) and try to understand driver functions.

FIR filter library

The library contains a FIR filter that performs convolution on the input PPG signal. The filter requires a list of coefficients that can be determined in Matlab.

- Import the source file (**fir_filter.c**) in CM4 (Core1) → Source Files
- Repeat the process for the header file (**fir_filter.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 (**fir_filter.c**) and try to understand the convolution process and how the input signal is filtered.

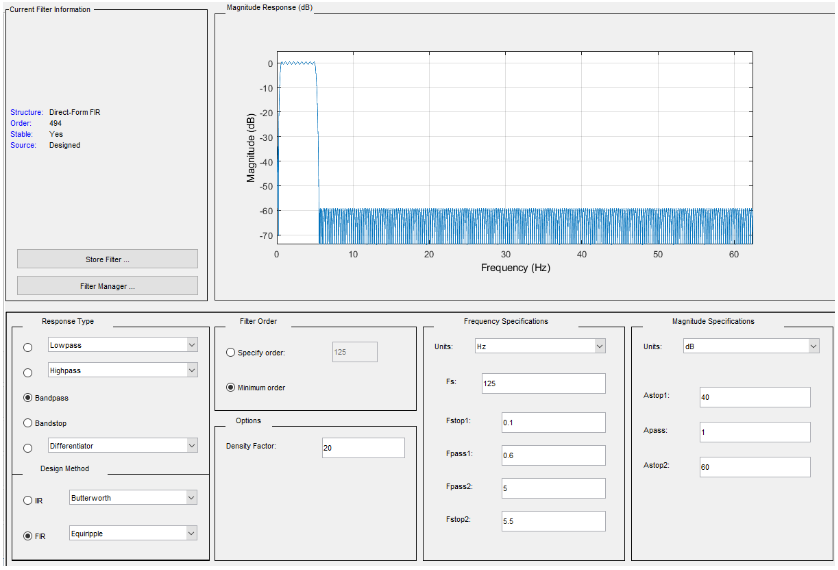
3 Implement digital filter

Many PPG applications use analog filtering modules to filter out undesired noise components. First and second order active band-pass filters are often used to filter DC components (with cutoff frequencies as low as 0.15 Hz) as well as high frequency noise signals (cutoff frequencies from 5Hz up to 60 Hz)

The SpO2 measurement is based on time multiplexing for red and infrared wavelengths. The LEDs must be switched on/off at a frequency range ranging from few hundreds to few thousand of Hz. The analog module that includes the high pass, low pass and the inverting amplifier impose a barrier on the switching frequency, since the overall time constant of the circuit is in the range of 1/2 seconds. Therefore, the signal should be acquired from the transimpedance amplifier and filtered in digital. In such a way, higher frequencies can be achieved for LED multiplexing.

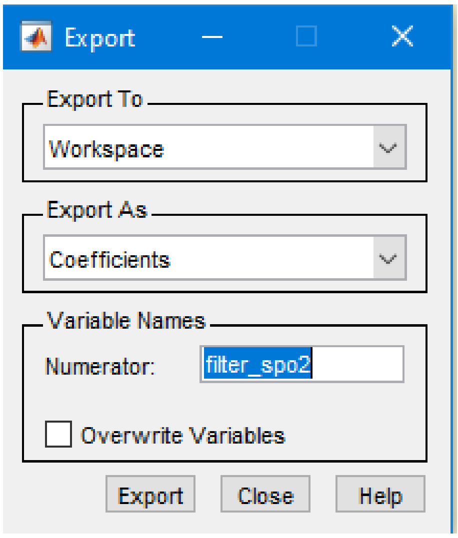
Matlab provides a signal processing toolbox that can be used to design digital filters. Filter Design and Analysis Tool (FDATool) allows the user to design, analyze and modify existing filter designs.

- Open Matlab and lunch the FDATool by executing *fdatool* command.
- Design a band pass filter as follows:

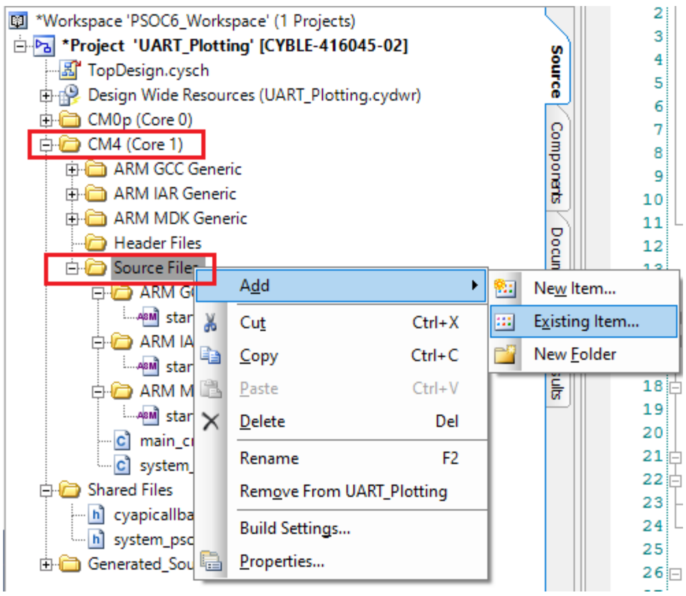


- The sampling frequency is defined by **TIMER_AdcRead_PERIOD_NSEC** in *custom_ppg.c* file. The value represents a sampling frequency of 125 Hz.

- Export filter coefficients (Ctrl + E)



- Add coefficients file (*.dat*) to the project



- Create a small Matlab script and export the coefficients to a *.dat* file

```
dlmwrite('band_pass_filter.dat', int32(filter_spo2 * 2^22), 'delimiter',',')
stem(int32(low_pass_10_12 * 2^22))
```

- Go to *fir_filter.c* and include the filter coefficients

```
void fir_filter (data_t *y, data_t x)
{
  const coef_t c[N+1] =
{
  #include "band_pass_filter.dat"
};

void fir_filter2 (data_t *y, data_t x)
{

  const coef_t c2[N2+1] =
{
  #include "band_pass_filter.dat"
};
```

4 SpO2 algorithms

To compute the R ratio, DC components and AC components of both signals (red, IR) have to be determined. This can either be achieved by analyzing the signal in time domain or by analyzing the signal in frequency domain.

- to be explained

5 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 SpO2. The values will be displayed on the OLED display for both sensors. 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
 *  =======================================================================================

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

/* @brief Include custom libraries for PPG EduKit */
#include "utils.h"
#include "MAX30102.h"
#include "milliseconds.h"
#include "spo2_algorithm.h"
#include "AD5273BRJZ1.h"
#include "TLC5925.h"
#include "custom_ppg.h"
#include "oled_driver.h"
#include "serial_frame.h"
#include "fir_filter.h"
```

- Declare local macros. These macro directives are used for MAX30102 configuration, as parameters for **MAX30102_Setup** function.

```c
/*==========================================================================================
 *                                          LOCAL MACROS
 *   ==========================================================================================*/
/* @brief Size of averaging buffer for HR and SpO2 measurements */
#define BPM_AVERAGE_RATE           (6U)
#define SPO2_AVERAGE_RATE          (6U)

/* @brief MAX30105 Options: 0=Off to 255=50mA */
#define CFG_LED_BRIGHTNESS    (60)
/* @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       (100)
/* @brief MAX30105 Options: 69, 118, 215, 411 */
#define CFG_PULSE_WIDTH       (411)
/* @brief MAX30105 Options: 2048, 4096, 8192, 16384 */
#define CFG_ADC_RANGE         (16384)
```

- Declare macro directives for LCD cursors

```c
/* @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        (90U)
```

- Declare global variables.

```c
/*==========================================================================================
 *                                        GLOBAL VARIABLES
 *   ==========================================================================================
/* @brief OLED custom text */
char oledText[][20] = {"MAX30105 SpO2: ", "CUSTOM SpO2: "};
/* @brief Buffer that stores the ASCII values of HR */
char numdec_buffer[USINT2DECASCII_MAX_DIGITS + 1];
/* @brief Stores the SpO2 value */
int32_t spo2;
/* @brief Stores the status of SpO2 value */
int8_t validSPO2;
/* @brief Stores the HR value */
int32_t heartRate;
/* @brief Stores the status of HR value */
int8_t validHeartRate;
/* @brief Temporary variables used for average */
float gTempSpO2Avg;
float gCustomTempSpO2Avg;
/* @brief SpO2 buffer and index used for averaging */
float gSpO2Buff[BPM_AVERAGE_RATE];
float gCustomSpO2Buff[BPM_AVERAGE_RATE];
uint8_t gSpO2CurrentIndex = 0U;
uint8_t gCustomSpO2CurrentIndex = 0U;
/* @brief Stores the status of the measurement */
bool gTissueDetected = FALSE;
/* @brief Buffers that store the data from the FIFO buffer */
int32_t PPG_bufferRed[CUSTOM_PPG_BUFFER_LENGTH];
int32_t PPG_bufferIR[CUSTOM_PPG_BUFFER_LENGTH];
```

- Declare local functions

```c
/*==========================================================================================
 *                                    LOCAL FUNCTION PROTOTYPES
 *   ==========================================================================================
static void clearSpO2Digits(void);
static void displaySpO2Text(void);
static void refreshSpO2Values(void);

```

- The LCD should be updated with new SpO2 values every second. Declare a function that clears the old Spo2 values.

```c
/***************************************************************************
 * Function: clearSpO2Digits
 *
 * Description: Prepare OLED region for new SpO2 values. Delete old values.
 ***************************************************************************/
static void clearSpO2Digits(void)
{
    for(uint8_t i = OLED_HR_VALUE_TEXT_X_POSITION; i < OLED_HR_VALUE_TEXT_X_POSITION + 40; 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();
}
```

- Declare a function that displays the custom message texts

```c
/***************************************************************************
 * Function: displaySpO2Text
 *
 * Description: Prepare OLED region for new SpO2 values. Delete old values.
 ***************************************************************************/
static void displaySpO2Text(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();
}
```

- Create a function that displays the SpO2 values for the sensor module and for the custom analog module. The value should be displayed with 2 decimal precision.

```c
/*****************************************************************************
 * Function: refreshSpO2Values
 *
 * Description: Display new SpO2 values on the OLED display
 *****************************************************************************/
static void refreshSpO2Values(void)
{
    int whole = gTempSpO2Avg;
    int reminder = (gTempSpO2Avg - whole) * 100;
    if(whole == 100U)
    {
      whole = 99;
      reminder = 99;
    }
    /* Clear old HR values */
    clearSpO2Digits();
    /* Prepare to display new MAX3010x SpO2 value */
    gfx_setCursor(OLED_HR_VALUE_TEXT_X_POSITION, OLED_MAX30105_TEXT_Y_POSITION);
    /* Convert dec number to ASCII value */
    display_usint2decascii(whole, 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]);
    }

    gfx_setCursor(OLED_HR_VALUE_TEXT_X_POSITION + 10, OLED_MAX30105_TEXT_Y_POSITION);
    gfx_write('.');
    gfx_setCursor(OLED_HR_VALUE_TEXT_X_POSITION + 17 , OLED_MAX30105_TEXT_Y_POSITION);

    display_usint2decascii(reminder, 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]);
    }
```

- Replicate the code inside the function to display the SpO2 value for the custom sensor

```c
    whole = gCustomTempSpO2Avg;
    reminder = (gCustomTempSpO2Avg - whole) * 100;

    if(whole == 100U)
    {
      whole = 99;
      reminder = 99;
    }
    /* Prepare to display new custom SpO2 value */
    gfx_setCursor(OLED_HR_VALUE_TEXT_X_POSITION, OLED_CUSTOM_TEXT_Y_POSITION);
    /* Convert dec number to ASCII value */
    display_usint2decascii(whole, 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]);
    }

    gfx_setCursor(OLED_HR_VALUE_TEXT_X_POSITION + 10, OLED_CUSTOM_TEXT_Y_POSITION);
    gfx_write('.');
    gfx_setCursor(OLED_HR_VALUE_TEXT_X_POSITION + 17 , OLED_CUSTOM_TEXT_Y_POSITION);

    display_usint2decascii(reminder, numdec_buffer);
    /* Draw the ASCII SpO2 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 afferent interrupt vector. Then, the interrupts are enabled and the UART peripheral is initialized.

  The first component that is initialized is the AD5273 digital potentiometer, which is used for setting the current for the LED driver. The **AD5273_Init** function initializes the I2C peripheral, sets the data rate and the clock frequency. The LED current is set to 5 mA by calling the **TLC5925_SetCurrent_mA** function. The function translates the current value to a 6 bit value that represents the digital potentiometer resistance and writes the value over I2C. Having the external resistance configured, the LED driver can be configured to enable the infrared LED by calling **TLC5925_enableIR**. The PPG signal can be measured right after the LED is turned on. Therefore, the ADC peripheral is started and the timer used to trigger the ADC conversion is initialized and started.

```c
int main(void)
{
    /* Assign ISR routines */
    MILLIS_AssignISR();
    CUSTOM_PPG_AssignISR_AdcRead();

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

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

    /* 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);

    /* Init digital potentiometer */
    AD5273_Init();
    /* Set 5 mA current for the LED driver */
    TLC5925_SetCurrent_mA(5);
    /* Enable IR LED */
    TLC5925_enableIR();
    /* Start ADC and ADC Conversion */
    ADC_Start();
    /* Start timer used for ADC reading */
    CUSTOM_PPG_InitAndStartTimer_AdcRead();

```

- The display is initialized and the custom texts are displayed. The MAX30102 sensor is initialized by calling the setup function with the desired configuration values. The parameters can take only few values and have been described above.

```c
    /* Wait for VCC stable before initializing the OLED display */
    CyDelay(1000);
    /* Init 128x64 OLED FeatherWing display. */
    for(uint8_t i =0; i<3; i++)
    {
        display_init();
        CyDelay(300);
    }
    CyDelay(1000);
    display_clear();
    display_update();
    /* Set display rotation to 1 (width -> height, height -> width) */
    gfx_setRotation(1);
    CyDelay(50);
    /* Display custom messages */
    displaySpO2Text();

    numdec_buffer[USINT2DECASCII_MAX_DIGITS] = '\0';

    /* Initialize MAX30105 sensor */
    MAX30105_Setup(CFG_LED_BRIGHTNESS, CFG_SAMPLE_AVERAGE, CFG_LED_MODE, CFG_SAMPLE_RATE, CFG_PULSE_WIDTH, CFG_ADC_RANGE);

```

- In the while loop, the MAX30102 sensor should be read calling **MAX30102_ReadChannels** function in a busy loop. The **FIFO_Buffer** is filled until the head pointer reaches the maximum buffer size. A sliding window is used between two consecutive channel readings (the for loop is used to shift the buffers old values). The SpO2 values are averaged, and the average buffer length is 6.

```c
    while(1)
    {
      /****** START MAX30101 ******/
      /* Shift signal - overlapping window */
      for (uint8_t i = FIFO_NUMBER_OF_SAMPLES - FIFO_NUMBER_OF_OVERLAPPING_SAMPLES; i < FIFO_NUMBER_OF_SAMPLES; i++)
      {

        FIFO_Buffer[((RED_CHANNEL * FIFO_NUMBER_OF_SAMPLES) +
        (i - (FIFO_NUMBER_OF_SAMPLES - FIFO_NUMBER_OF_OVERLAPPING_SAMPLES)))] = FIFO_Buffer[((RED_CHANNEL * FIFO_NUMBER_OF_SAMPLES) + i)];
        FIFO_Buffer[((IR_CHANNEL * FIFO_NUMBER_OF_SAMPLES) +
        (i - (FIFO_NUMBER_OF_SAMPLES - FIFO_NUMBER_OF_OVERLAPPING_SAMPLES)))] = FIFO_Buffer[((IR_CHANNEL * FIFO_NUMBER_OF_SAMPLES) + i)];

      }
      samplesTaken = FIFO_NUMBER_OF_OVERLAPPING_SAMPLES;
      /* Pool sensor until the number of samples is equal to FIFO_NUMBER_OF_SAMPLES */
      while(FIFO_NUMBER_OF_SAMPLES != samplesTaken)
      {
        /* Delay to reduce noise over custom PPG signal due to I2C noise coupling */
        CyDelay(50);
        gTissueDetected = MAX30105_ReadChannels(&FIFO_Buffer[0], TRUE);
      }
      /* Compute HR and SpO2 */
      maxim_heart_rate_and_oxygen_saturation(&FIFO_Buffer[IR_CHANNEL * FIFO_NUMBER_OF_SAMPLES], FIFO_NUMBER_OF_SAMPLES,
                                             &FIFO_Buffer[RED_CHANNEL * FIFO_NUMBER_OF_SAMPLES],
                                             &spo2, &validSPO2, &heartRate, &validHeartRate,
                                             CFG_SAMPLE_RATE/CFG_SAMPLE_AVERAGE, MAX30105_SAMPLES_BETWEEN_PEAKS);
     if(TRUE == validSPO2)
     {
        gSpO2Buff[gSpO2CurrentIndex++] = spo2;
        gSpO2CurrentIndex %= SPO2_AVERAGE_RATE;
        gTempSpO2Avg = 0;
        for (uint8_t x = 0 ; x < SPO2_AVERAGE_RATE; x++)
          gTempSpO2Avg += gSpO2Buff[x];
        gTempSpO2Avg /= SPO2_AVERAGE_RATE;
     } /****** END MAX30101 ******/
```

- The same reading process is done for the custom analog module. The buffers **CUSTOM_PPG_bufferIR** and **CUSTOM_PPG_bufferRed** are filled inside the **isr_adcRead** routine. When the buffer is full, the buffers are copied internally and the flag **bBufferProcessed** is set. For signal is filtered for each length and then the SpO2 function is called. The SpO2 values are averaged, and the average buffer length is 6. Finally, the new SpO2 values are updated on the LCD.

```c
     if(CUSTOM_PPG_BUFFER_LENGTH == CUSTOM_PPG_bufferHead) /****** START CUSTOM SOLUTION ******/
     {
        memcpy(PPG_bufferRed, (int32_t *) CUSTOM_PPG_bufferRed, sizeof(PPG_bufferRed));
        memcpy(PPG_bufferIR, (int32_t *) CUSTOM_PPG_bufferIR, sizeof(PPG_bufferIR));
        /* Shift signal - overlapping window */
        for (uint16_t i = CUSTOM_PPG_BUFFER_LENGTH - CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES; i < CUSTOM_PPG_BUFFER_LENGTH; i++)
        {
            CUSTOM_PPG_bufferIR[(i - (CUSTOM_PPG_BUFFER_LENGTH - CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES))] = CUSTOM_PPG_bufferIR[i];
            CUSTOM_PPG_bufferRed[(i - (CUSTOM_PPG_BUFFER_LENGTH - CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES))] = CUSTOM_PPG_bufferRed[i];
        }
        bBufferProcessed = TRUE;
        CUSTOM_PPG_bufferHead = CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES;
        for(uint16_t i = 0; i < CUSTOM_PPG_BUFFER_LENGTH; i++)
        {
           fir_filter((data_t *)&PPG_bufferIR[i], (data_t) PPG_bufferIR[i]);
           fir_filter2((data_t *)&PPG_bufferRed[i], (data_t) PPG_bufferRed[i]);
        }
        maxim_heart_rate_and_oxygen_saturation2((int32_t *)&PPG_bufferIR[0], CUSTOM_PPG_BUFFER_LENGTH,
                                                (int32_t *)&PPG_bufferRed[0], &spo2, &validSPO2,
                                                 &heartRate, &validHeartRate,
                                                 CUSTOM_PPG_FREQS, CUSTOM_SAMPLES_BETWEEN_PEAKS);
        if(TRUE == validSPO2)
        {
            gCustomSpO2Buff[gCustomSpO2CurrentIndex++] = spo2;
            gCustomSpO2CurrentIndex %= SPO2_AVERAGE_RATE;
            gCustomTempSpO2Avg = 0;
            for (uint8_t x = 0 ; x < SPO2_AVERAGE_RATE; x++)
              gCustomTempSpO2Avg += gCustomSpO2Buff[x];
            gCustomTempSpO2Avg /= SPO2_AVERAGE_RATE;
        }/****** END CUSTOM SOLUTION ******/
        /* Refresh OLED display with new SpO2 values */
        refreshSpO2Values();
     }
   }
}
```

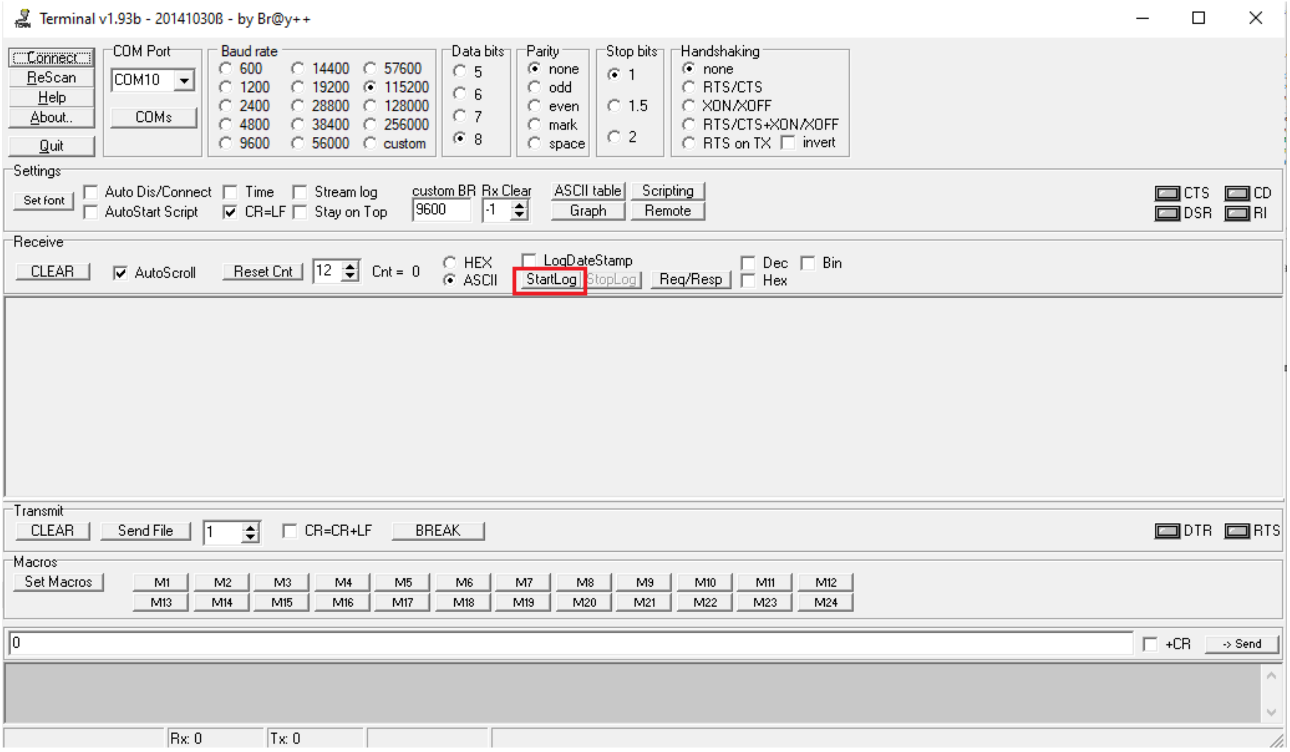
6 Compute SpO2 in Matlab

To process the data in Matlab, first record the data and send it over UART.

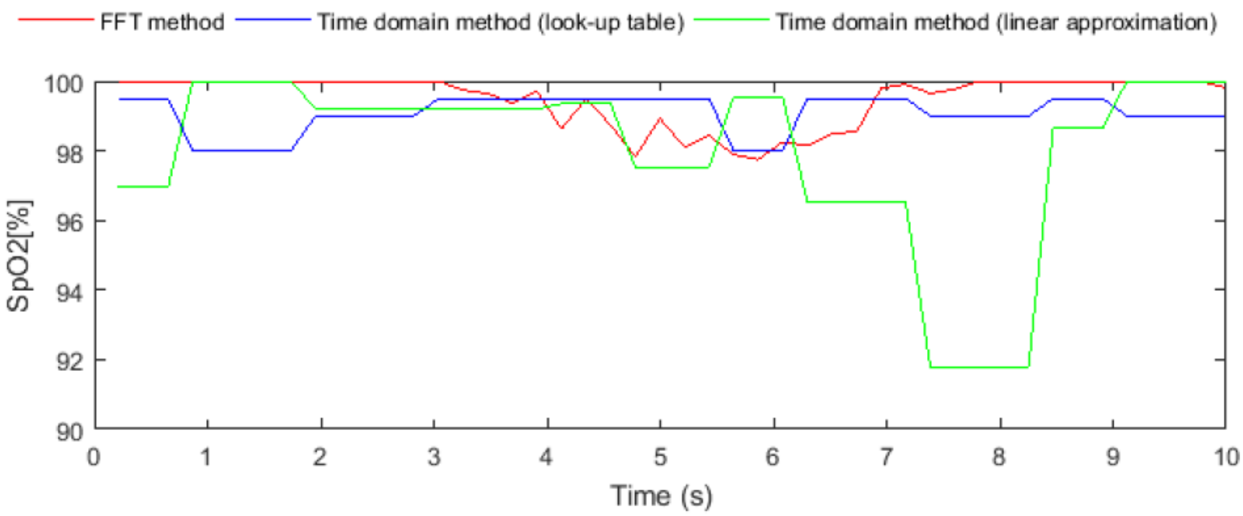
- Insert a loop in the main function to print the IR and red buffers over serial.

```c
        for(uint16_t i = 0; i < CUSTOM_PPG_BUFFER_LENGTH; i++)
        {
            fir_filter((data_t *)&PPG_bufferIR[i], (data_t) PPG_bufferIR[i]);
            fir_filter2((data_t *)&PPG_bufferRed[i], (data_t) PPG_bufferRed[i]);
        }

        for (uint16_t i = CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES; i < CUSTOM_PPG_BUFFER_LENGTH; i++)
        {
             printf("%i %i \n\r", PPG_bufferRed[i], PPG_bufferIR[i]);
             CyDelay(10);
        }

```

- Use any serial terminal to read and store the data. For instance, one can use Terminal v1.93b.



- Import the log file in the Matlab workspace and run the **PPG_SpO2_Algorithms.m** script. The output should be a graph that plots the SpO2 values for each method.



- Play with window size and with the overlapping ratio to see the impact on SpO2 measurements.

References

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

[2] 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

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

[4] https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library

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