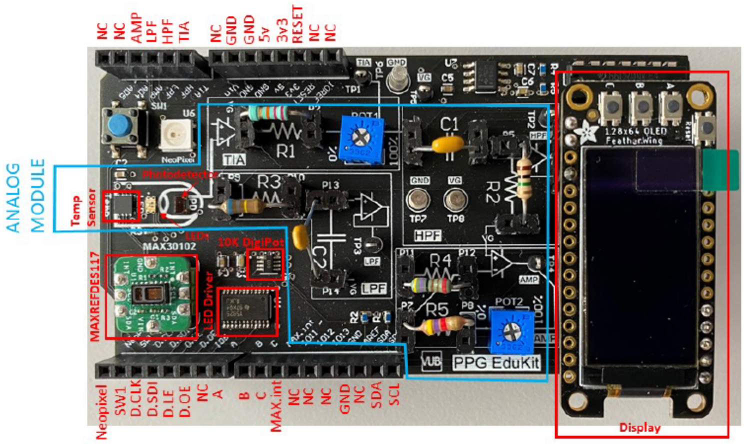
Lab 4: Compare MAX30102 PPG sensor and custom analog PPG module (PSOC6)

Introduction

This lab helps the user to acquire the PPG signal using two different sensor setups: using the analog conditioning stages which uses one photodetector and three different wavelengths (green, red and IR) or using the commercial MAXREFDES117 PPG module that includes the MAX30102 sensor with only two wavelengths (red and IR).

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

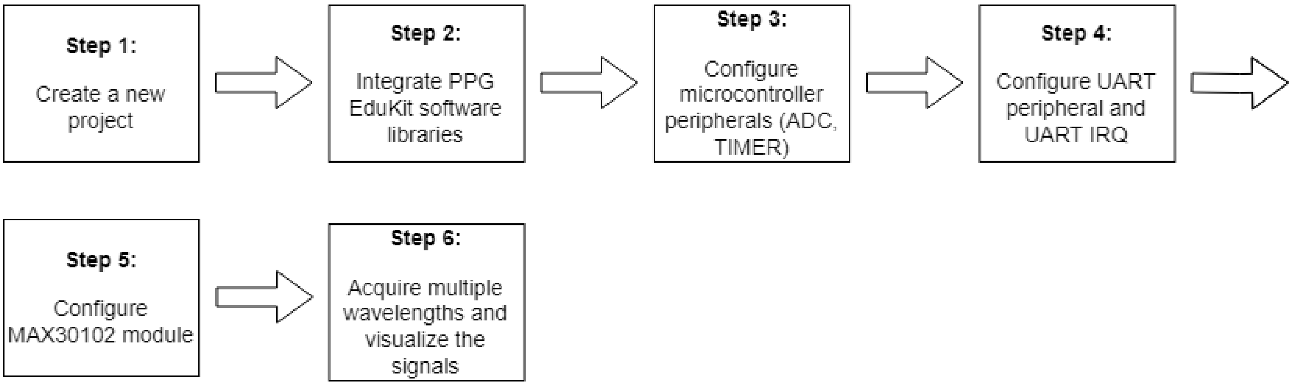
After completing this lab, you will be able to:

- Understand how to set up a PSOC Creator project
- Use and modify the PPG EduKit software libraries
- Configure microcontroller peripherals in PSOC Creator (ADC, Timer, IRQ, UART)
- Understand multiwavelength measurement and control acquisition wavelength
- Read data from MAX30102 sensor
- Use the PPG EduKit GUI application and visualize multiple wavelength signals

Procedure

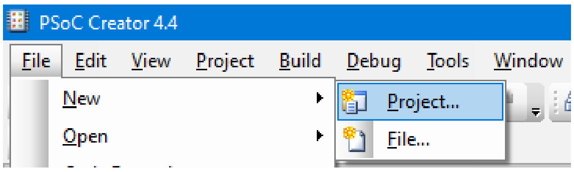
This lab is separated into steps that provide information on the detailed instructions that follow. Follow these detailed instructions to progress through the lab.

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, send and receive data over UART, convigure MAX30102 sensor, and finally run the PPG EduKit GUI application to visualize data and control acquisition wavelength

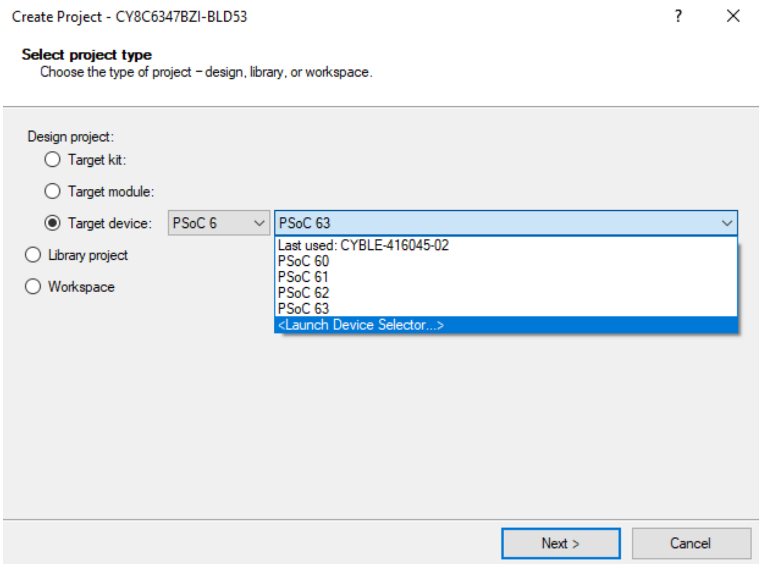


1 Creation of the Project

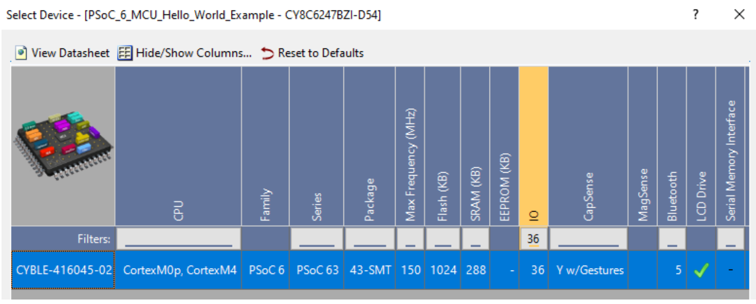
- Open PSoC Creator
- Go to File → New → Project



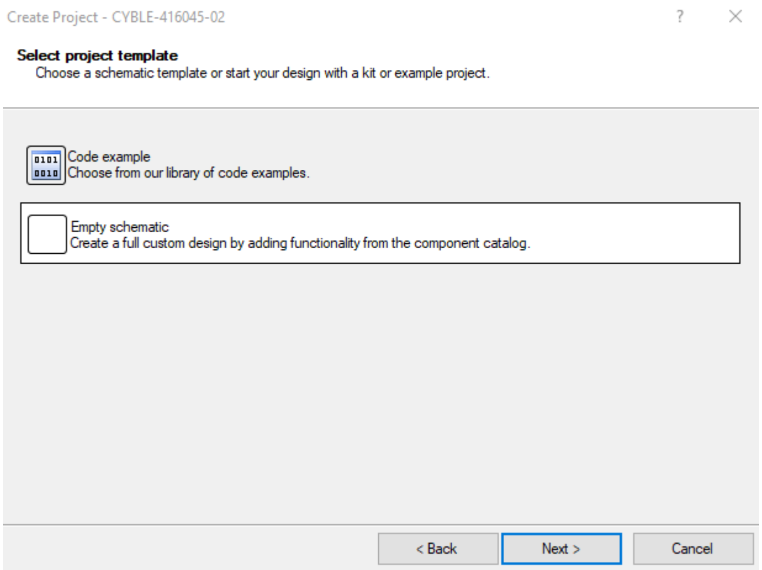
- Select the target device → PSOC6 → <Launch Device Selector>



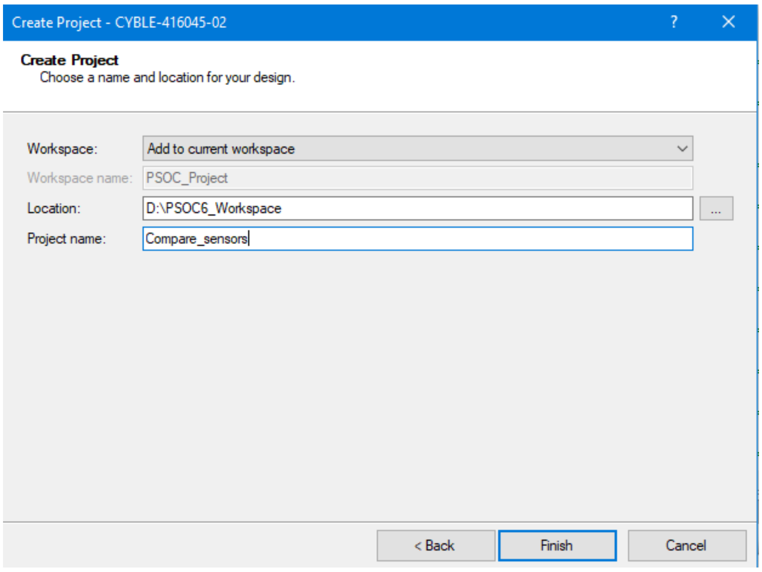
- Select the **CYBLE-416045-2** Device



- Select "Empty schematic" and click "Next"



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



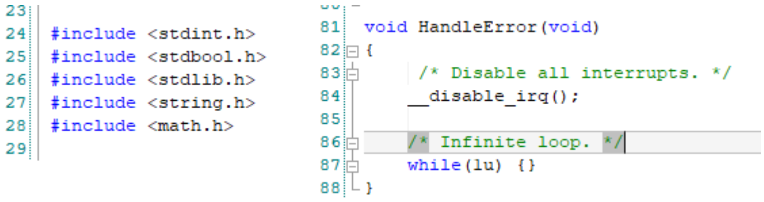
2 Integrate PPG EduKit software libraries

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.

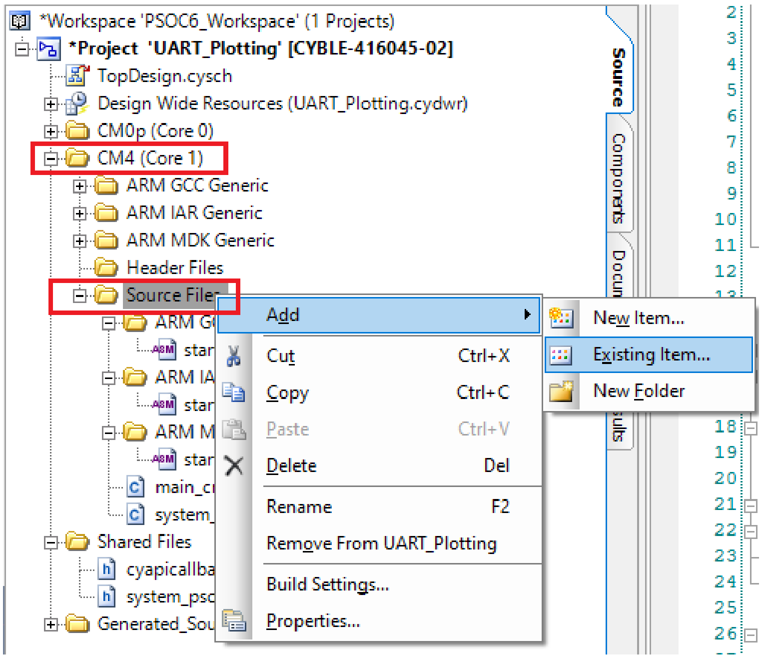
In order to plot the PPG signal using the PPG EduKit GUI the user has to integrate the following libraries: TLC5925 (LED driver library), AD5273 (digital potentiometer library), custom PPG, serial frame (UART), MAX30102 module, milliseconds and utils library. Each library consists of a source file (.c) 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.



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

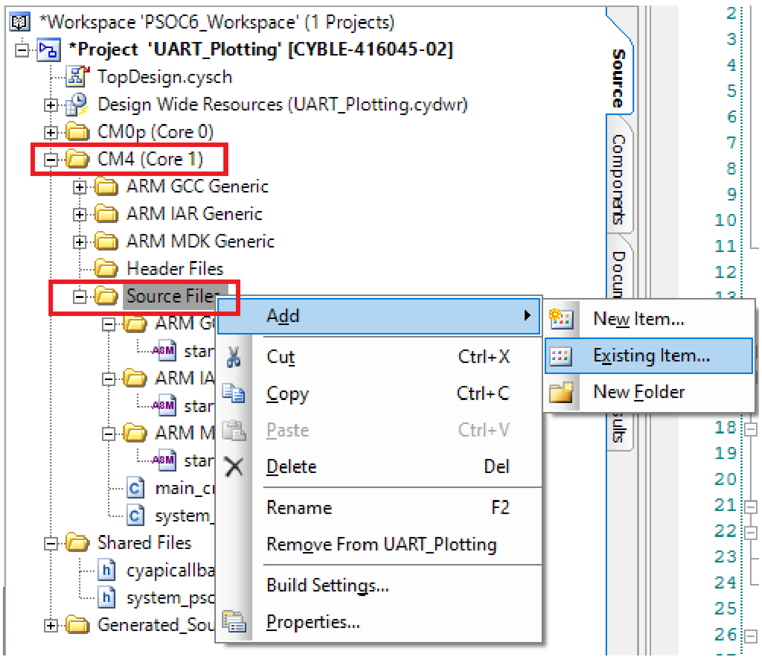


- Repeat the process for the header file (utils.h). Import the file in CM4 (Core1) → Header Files

AD5273 driver

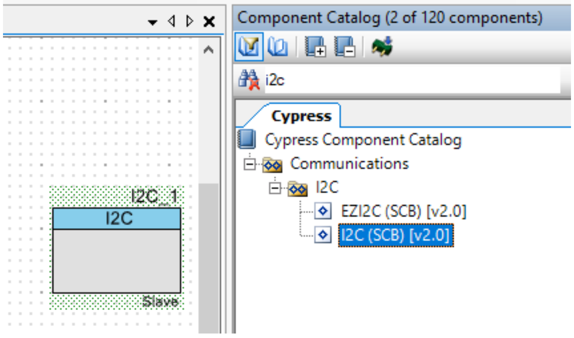
AD5273 is a 64-position, one-time programmable (OTP) digital potentiometer that employs fuse link technology to achieve permanent program setting and can be configured through an I2C-compatiblle interface.

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

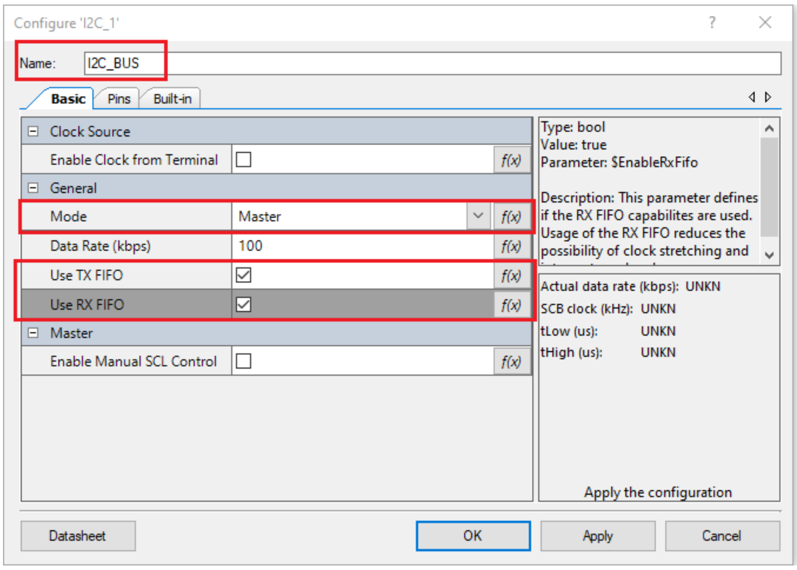


- Repeat the process for the header file (AD5273.h). Import the file in CM4 (Core1) → Header Files

- 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.

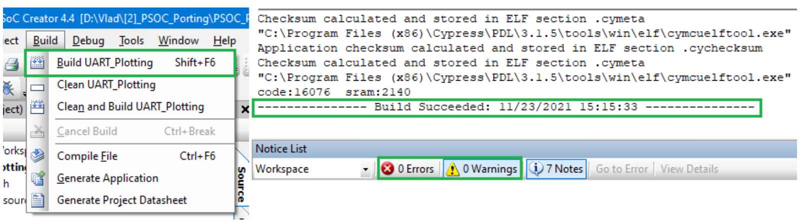


- Configure the I2C component by double click on it. Rename the component as **I2C_BUS**, set the mode as **master** and enable **TX/RX FIFO** buffers.



- 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.

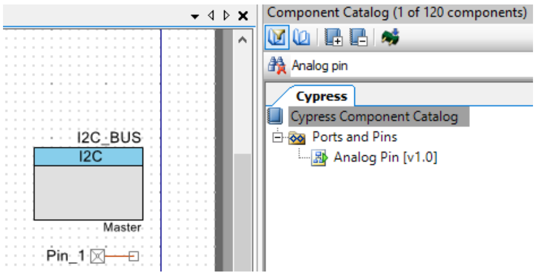


- Take a look at the application programming interface (AD5273.c) and try to understand driver functions.

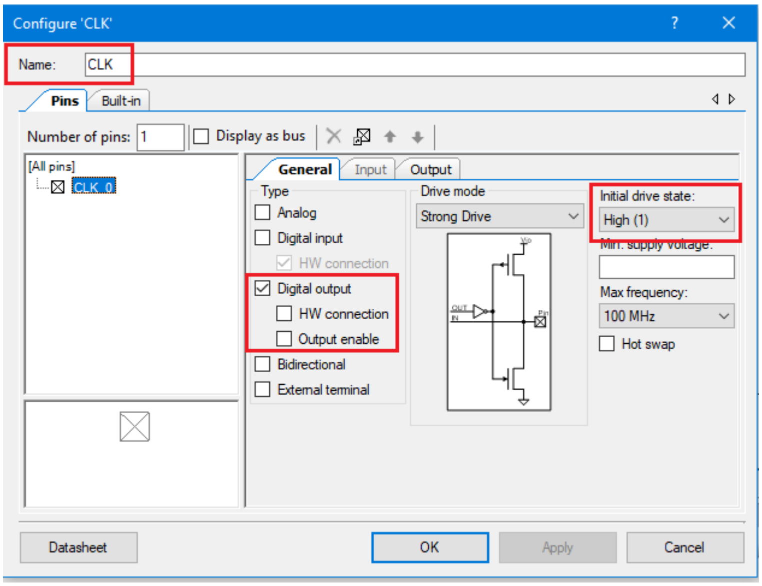
TLC5925 driver

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.

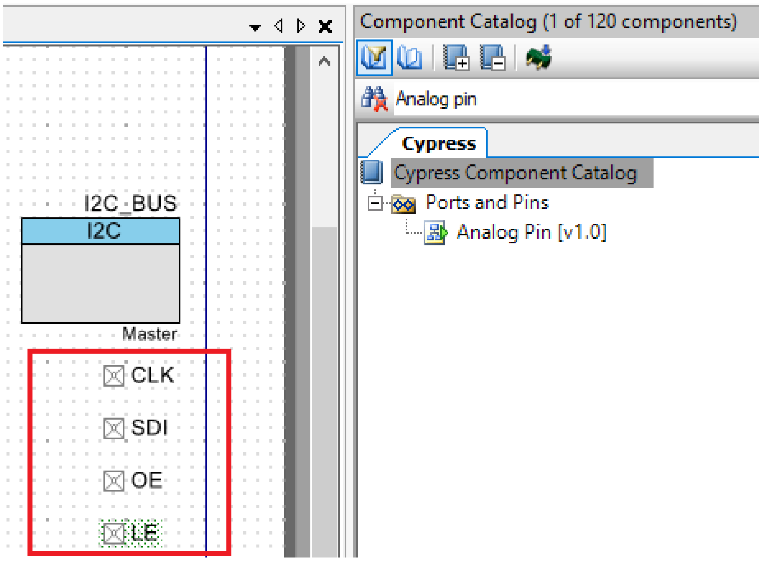
- Import the source file (TLC5925.c) in CM4 (Core1) → Source Files
- Repeat the process for the header file (TLC5925.h). Import the file in CM4 (Core1) → Header Files
- Go to the **TopDesign** schematic. In the component catalog (right panel at the right of the screen), write **Analog pin** and drag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



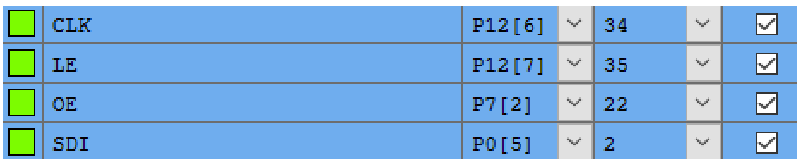
- Configure the Pin component by double click on it.



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

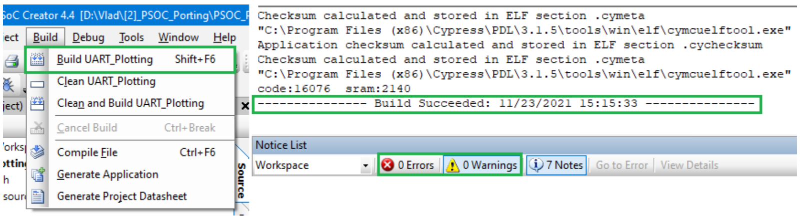


- Go to Design Wide Resources → Pins and assign the pins as follows:



	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.

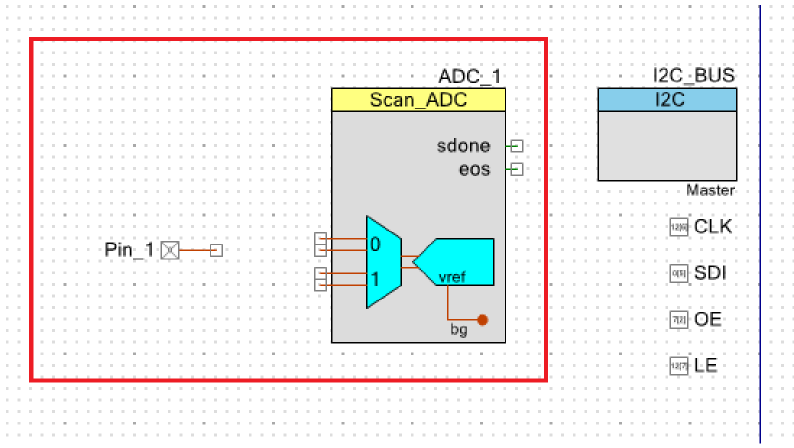


- Take a look at the application programming interface (TLC5925.c) and try to understand driver functions.

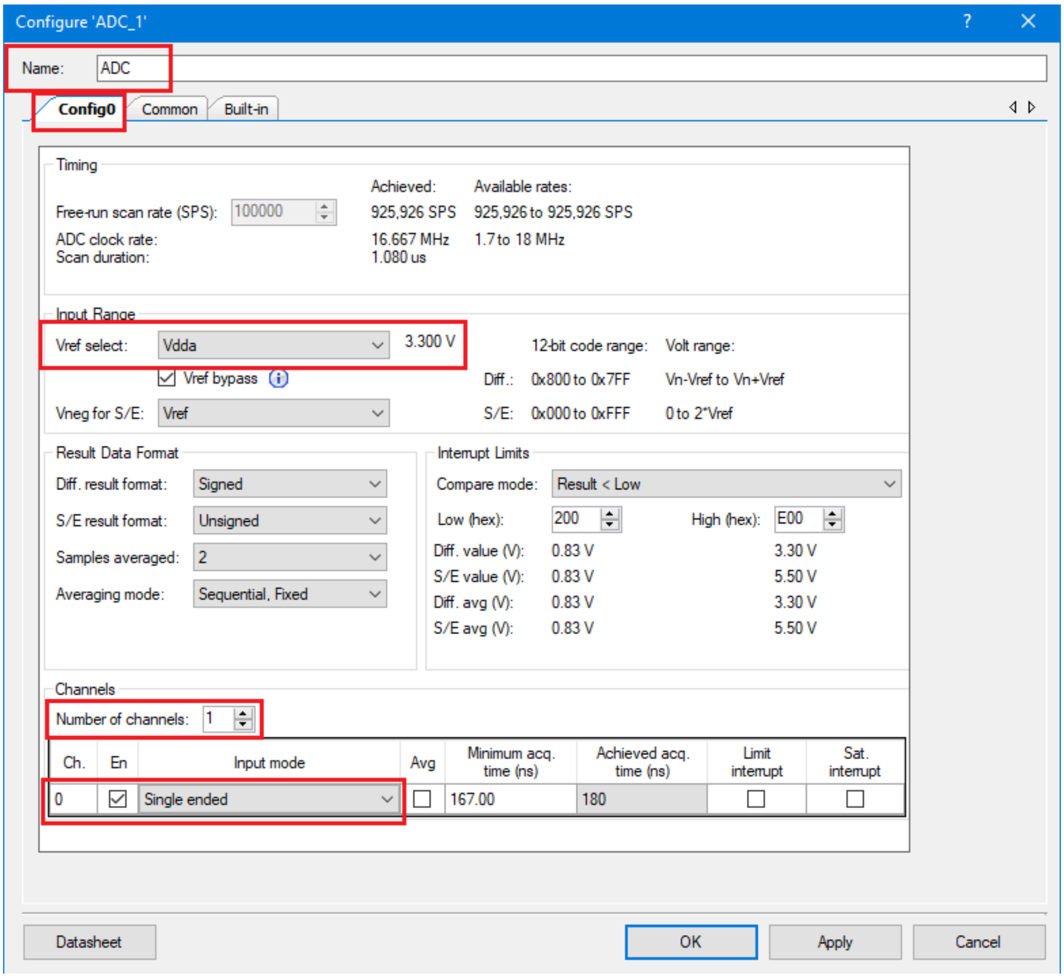
Custom PPG driver

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.

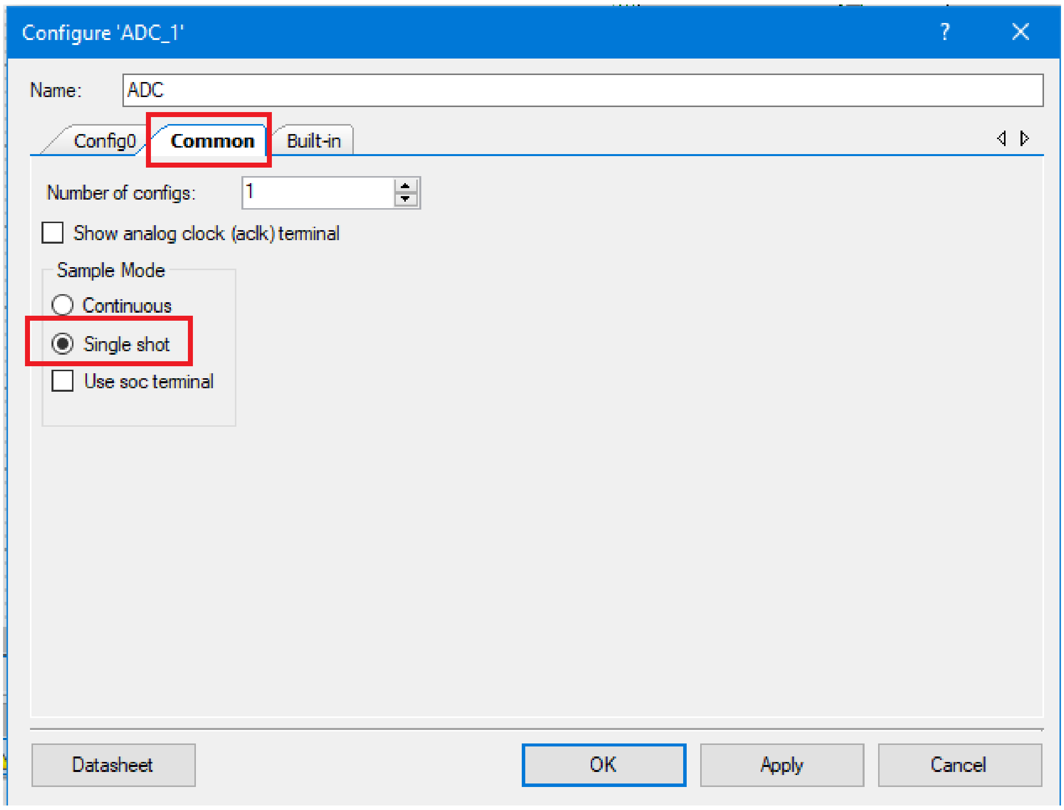
- Go to the **TopDesign** schematic. In the component catalog (right panel at the right of the screen), write **ADC** and drag and drop the component into the schematic. Search for **Analog pin** and drag and drop the component into the schematic.



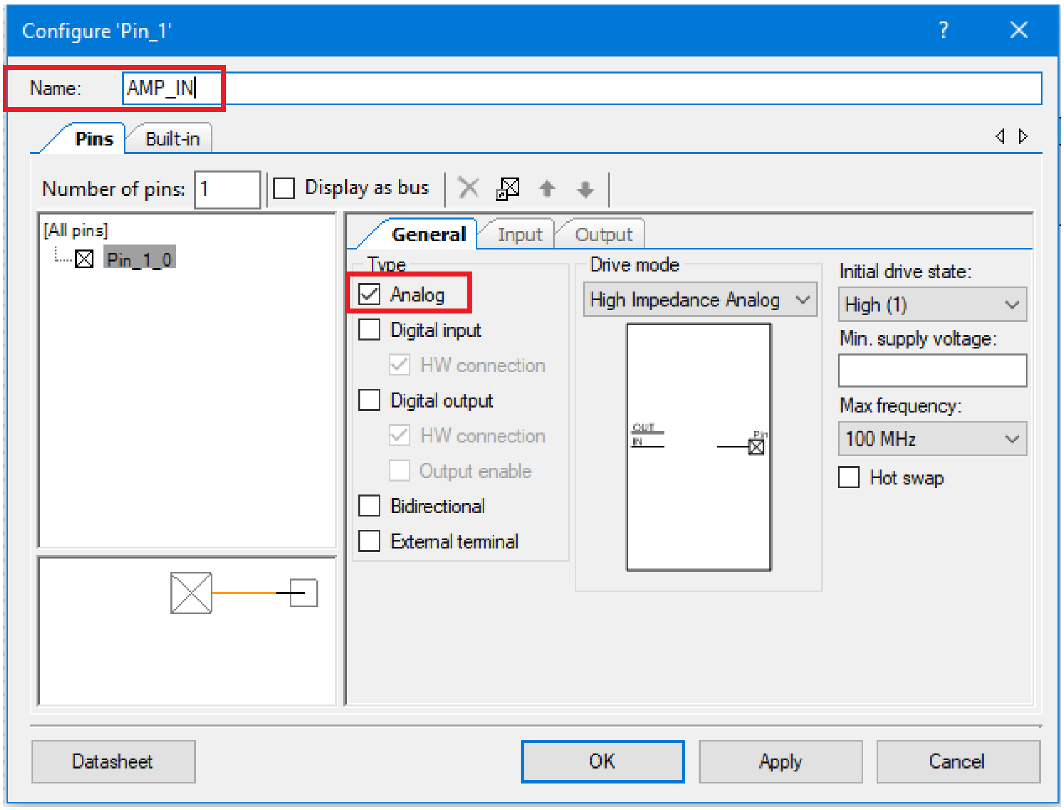
- Configure the ADC component by double click on it.



- Configure the ADC sample mode as single shot mode.



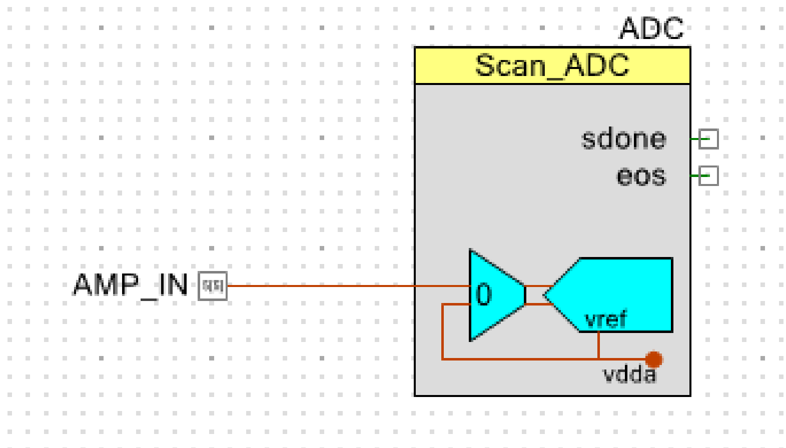
- Configure the Pin component by double click on it.



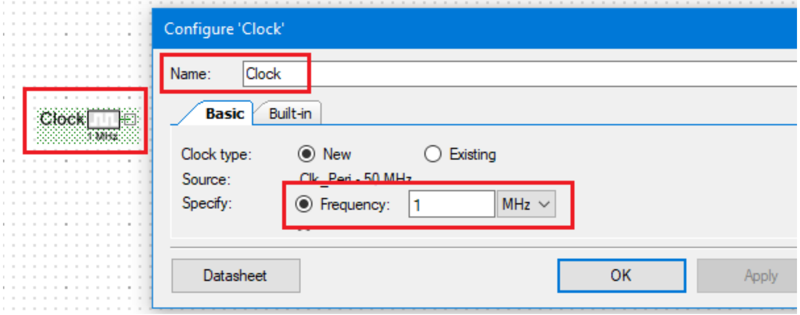
- Go to Design Wide Resources → Pins and assign the **AMP_IN** pin as follows:

	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	☑

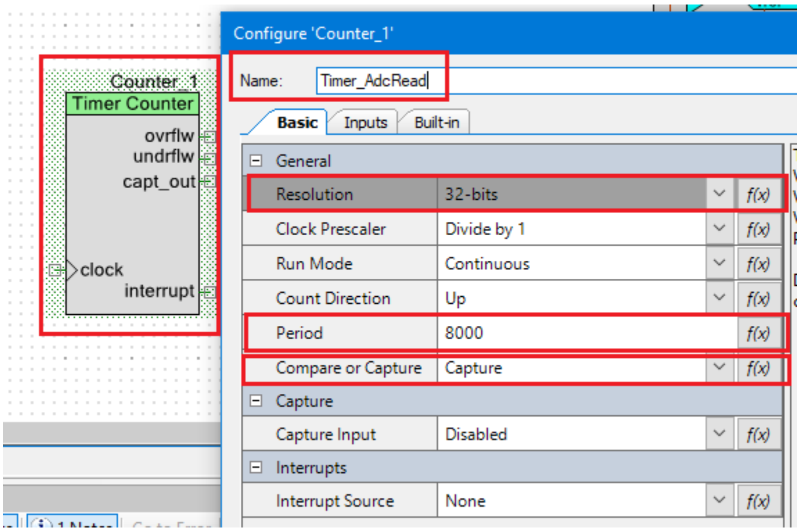
- Connect the analog pin to the ADC (press W to place a wire).



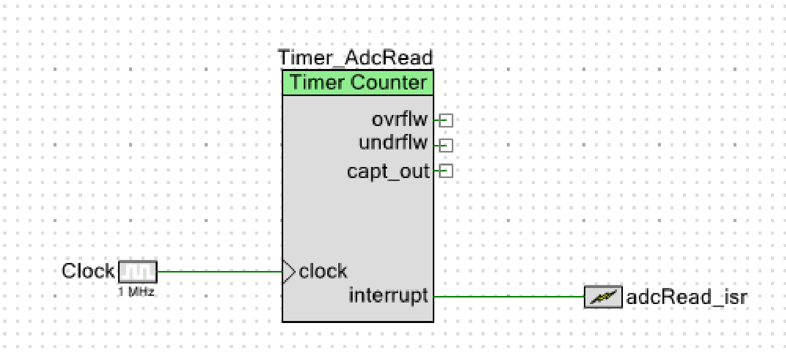
- Search for **Clock** and drag and drop the component into the schematic.
- Configure clock to 1 MHz



- 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

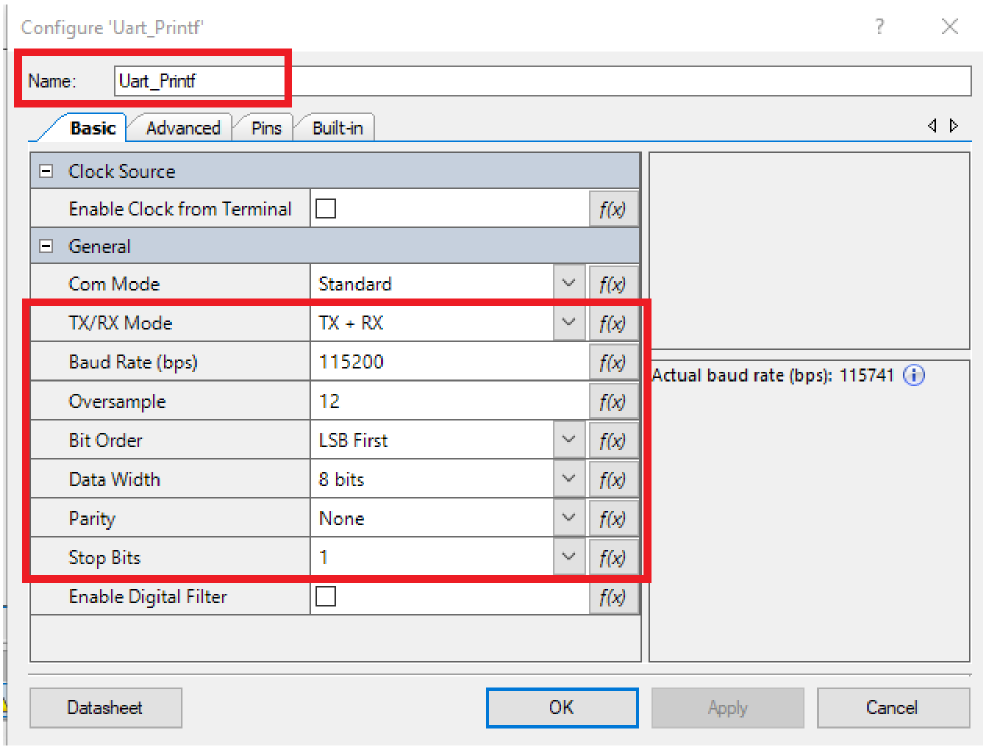


- Import the source file (custom_ppg.c) in CM4 (Core1) → Source Files
- Repeat the process for the header file (custom_ppg.h). Import the file in CM4 (Core1) → Header Files
- Build the project to check if there is any error.
- Take a look at the application programming interface (custom_ppg.c) and try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

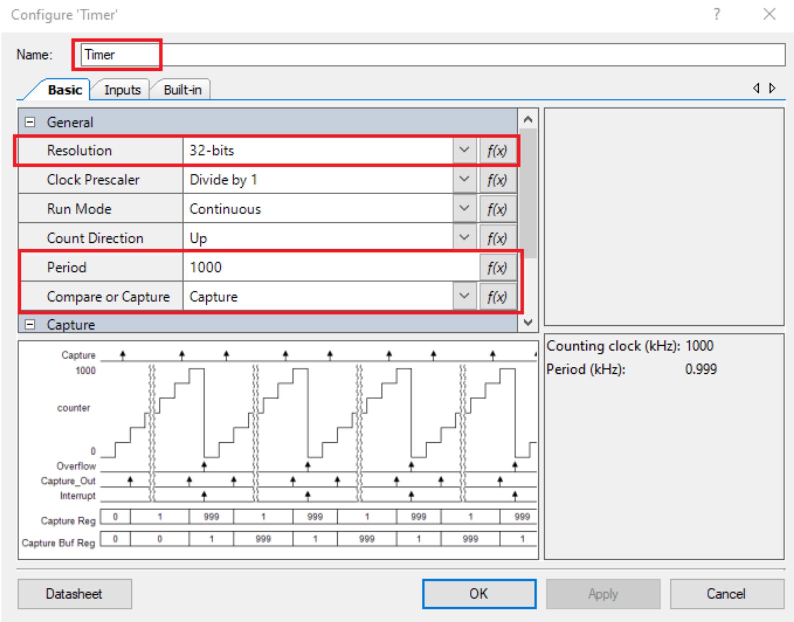
\Uart_Printf:rx\	P5[0]	39	☐
\Uart_Printf:tx\	P5[1]	40	☐

- 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.

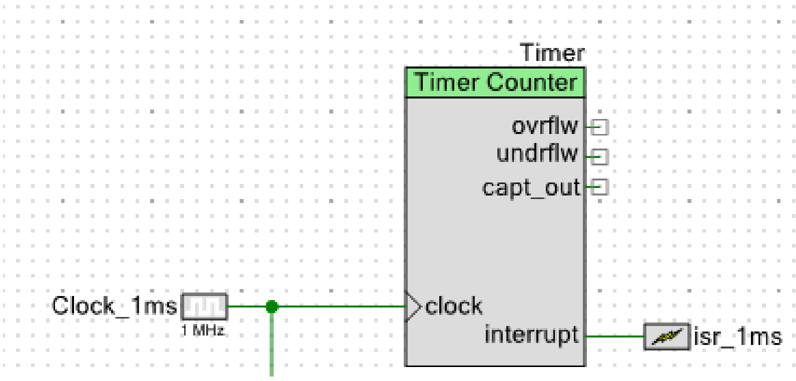
Milliseconds library

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

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 send it over UART. Also, the user can select the LED wavelength. 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 "AD5273.h"
#include "TLC5925.h"
#include "custom_ppg.h"
#include "serial_frame.h"
#include "MAX30102.h"
```

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

```

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

/* @brief MAX30102 Options: 0=Off to 255=50mA */
#define CFG_LED_BRIGHTNESS    (60)
/* @brief MAX30102 Options: 1, 2, 4, 8, 16, */
#define CFG_SAMPLE_AVERAGE    (4)
/* @brief MAX30102 Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green */
#define CFG_LED_MODE          (2)
/* @brief MAX30102 Options: 50, 100, 200, 400, 800, 1000, 1600, 3200 */
#define CFG_SAMPLE_RATE       (3200)
/* @brief MAX30102 Options: 69, 118, 215, 411 */
#define CFG_PULSE_WIDTH       (411)
/* @brief MAX30102 Options: 2048, 4096, 8192, 16384 */
#define CFG_ADC_RANGE         (16384)

```

- Declare the global variables. The serial frame command is represented by the **frameParams_t** structure type. The variable is passed to the **createSerialFrame** function in order to create the desired data frame. The function returns the address location of the data frame buffer to be sent, and **gpFrame** pointer is used to point to that memory location. Receiving data from UART is done using an interrupt request. Few variables have to be declared for this purpose. **Power**$_{l}evelvariableisusedtostorethecurrentvalueforMAX30102sensor$.

```c
/*==========================================================================================
 *                                    GLOBAL VARIABLES
 *  ========================================================================================*/

/* @brief Serial frame command */
frameParams_t frameParam;

/* @brief Pointer to a serial frame newly created*/
uint8_t *gpFrame = NULL;

/* @brief Variables used for UART reading inside ISR */
volatile uint8_t rx_data[SERIAL_FRAME_LENGTH_MAX] = {0};
volatile uint32_t data_rx_index = 0;
volatile bool frameReceived = FALSE;
volatile uint8_t uartError;

/* @brief Variable used to set current for MAX30102 */
uint8_t power_level;
```

- Before the main function declaration, **ISR_UART** function has to be declared. This function is later linked as the ISR handler for every RX interrupt request. Inside the function it is checked if the source of interrupt is due to "FIFO not empty". The interrupt flag is cleared and the data is read into **rx_data** buffer. The data is added to the buffer until the frame terminator is detected, then the buffer index is reset and the **frameReceived** flag is set as true.

```c
/*==========================================================================================
 *                                    GLOBAL FUNCTIONS
 *  ========================================================================================*/
void ISR_UART(void)
{
    /* Check for "RX fifo not empty interrupt" */
    if((Uart_Printf_HW->INTR_RX_MASKED & SCB_INTR_RX_MASKED_NOT_EMPTY_Msk ) != 0)
    {
        /* Clear UART "RX fifo not empty interrupt" */
        Uart_Printf_HW->INTR_RX = Uart_Printf_HW->INTR_RX & SCB_INTR_RX_NOT_EMPTY_Msk;
        /* Get the character from terminal */
        rx_data[data_rx_index] = Cy_SCB_UART_Get(Uart_Printf_HW);

        /* Update data_received index */
        if(rx_data[data_rx_index] == FRAME_TERMINATOR_2)
        {
            data_rx_index = 0;
            frameReceived = TRUE;
        }
        else
        {
            frameReceived = FALSE;
            data_rx_index++;
        }
    }
    else
    {
        uartError = 1;
    }
}
```

- The main function starts with the initialization of **adcRead_isr** interrupt by setting the priority and the afferent interrupt vector. Then, the interrupts are enabled and the UART peripheral is initialized. Reading data from PC is done by configuring the UART RX FIFO interrupt and setting **ISR_UART** function as the handler.

  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.

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
int main(void)
{
    /* Assign ISR routines */
    CUSTOM_PPG_AssignISR_AdcRead();
    /* Enable global interrupts. */
    __enable_irq();

    /* 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);

    /* Unmasking only the RX fifo not empty interrupt bit */
    Uart_Printf_HW->INTR_RX_MASK = SCB_INTR_RX_MASK_NOT_EMPTY_Msk;
    /* Interrupt Settings for UART */
    Cy_SysInt_Init(&Uart_Printf_SCB_IRQ_cfg, ISR_UART);
    /* Enable the interrupt */
    NVIC_EnableIRQ(Uart_Printf_SCB_IRQ_cfg.intrSrc);


    /* 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();
    MAX30102_Setup(CFG_LED_BRIGHTNESS, CFG_SAMPLE_AVERAGE, CFG_LED_MODE,
                   CFG_SAMPLE_RATE, CFG_PULSE_WIDTH, CFG_ADC_RANGE);
```

- In the while loop, the **frameReceived** flag is checked every iteration. If a **SET_WAVELENGTH** frame is received, the value of the desired current value and wavelength are set. The current value for MAX30102 has to be mapped to a value between 5 and 255 (0.4 - 50 mA).

```c
    /* Main infinite loop */
    while(1)
    {
        if(frameReceived == TRUE)
        {
            frameReceived = FALSE;
            if (rx_data[1] == SET_WAVELENGTH)
            {
                power_level = (255 / 45) * (rx_data[3] - 5);
                activeChannel = rx_data[2];

                if(rx_data[3] > 40)
                {
                    TLC5925_SetCurrent_mA(40);
                }
                else
                {
                    TLC5925_SetCurrent_mA(rx_data[3]);
                }
                switch(activeChannel)
                {
                   case IR_CHANNEL:
                   TLC5925_enableIR();
                   MAX30102_SetPulseAmplitudeIR(power_level);
                   break;
                   case RED_CHANNEL:
                   TLC5925_enableRed();
                   MAX30102_SetPulseAmplitudeRed(power_level);
                   break;
                   case GREEN_CHANNEL:
                   TLC5925_enableGreen();
                   /* no green channel for MAX30102 (only for MAX30101) */
                   MAX30102_SetPulseAmplitudeGreen(power_level);
                   break;
                }
            }
        }
```

- 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). Sending the entire buffer over UART leads to delay in displaying the signals, due to large frame sizes. Only the new acquired samples are sent over UART and plotted.

```c
        for (uint8_t i = FIFO_NUMBER_OF_SAMPLES - FIFO_NUMBER_OF_OVERLAPPING_SAMPLES; i < FIFO_NUMBER_OF_SAMPLES; i++)
        {

            FIFO_Buffer[((activeChannel * FIFO_NUMBER_OF_SAMPLES) + (i - (FIFO_NUMBER_OF_SAMPLES - FIFO_NUMBER_OF_OVERLAPPING_SAMPLES)))]
                                = FIFO_Buffer[((activeChannel * 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)
        {
            MAX30102_ReadChannels(&FIFO_Buffer[0], TRUE);
        }

        /* Prepare serial frame structure */
        memset(&frameParam, 0x00, sizeof(frameParam));
        frameParam.frameType = CHANNEL_DATA;
        frameParam.sensor = MAX_SENSOR;
        frameParam.params.wavelength = activeChannel;
        frameParam.tissueDetected = TRUE;
        gpFrame = createSerialFrame(&FIFO_Buffer[(activeChannel * FIFO_NUMBER_OF_SAMPLES) + FIFO_NUMBER_OF_OVERLAPPING_SAMPLES],
                                    (FIFO_NUMBER_OF_SAMPLES - FIFO_NUMBER_OF_OVERLAPPING_SAMPLES) * 2, &frameParam);
        /* Sens serial frame */
        sendFrame(gpFrame);

    }
}

/* [] END OF FILE */
```

- Depending on the MAX30102 sampling rate, the **MAX30102_ReadChannels** call can take some time to execute, since the function pools the sensor for new data over I2C. This is why the data acquired using the custom sensor is sent over UART using an interrupt request.

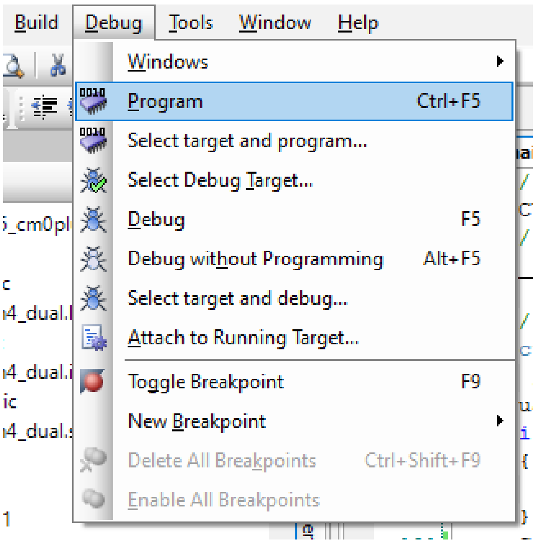
- In **custom_ppg.c** the interrupt handler for ADC reading is declared. The interrupt handler starts with clearing the interrupt flag. In the first if branch, the conversion is started and one of the buffers is filled with new data. It should be noticed that the I2C communication is interrupted before starting the ADC reading. It has been seen that this improves signal quality, since the I2C lines generates noise that is coupled into the analog signals.

```c
void CUSTOM_PPG_InterruptHandler_AdcRead(void)
{
    /* Clear the terminal count interrupt */

    Cy_TCPWM_ClearInterrupt(Timer_AdcRead_HW, Timer_AdcRead_CNT_NUM, CY_TCPWM_INT_ON_TC);


    if((CUSTOM_PPG_BUFFER_LENGTH != CUSTOM_PPG_bufferHead) && (TRUE == bBufferProcessed))
    {
        if(I2C_BUS_MasterGetStatus() != CY_SCB_I2C_MASTER_BUSY)
        {
            I2C_BUS_MasterAbortWrite();
            I2C_BUS_MasterAbortRead();
        }

        ADC_StartConvert();
        if(ADC_IsEndConversion(CY_SAR_RETURN_STATUS) != 0)
        {
            switch(activeChannel)
            {
                case IR_CHANNEL:
                CUSTOM_PPG_bufferIR[CUSTOM_PPG_bufferHead] =  ADC_GetResult16(ADC_CHANNEL_0_INV_AMP);
                break;
                case RED_CHANNEL:
                CUSTOM_PPG_bufferRed[CUSTOM_PPG_bufferHead] =  ADC_GetResult16(ADC_CHANNEL_0_INV_AMP);
                break;
                case GREEN_CHANNEL:
                CUSTOM_PPG_bufferGreen[CUSTOM_PPG_bufferHead] =  ADC_GetResult16(ADC_CHANNEL_0_INV_AMP);
                break;
            }

            CUSTOM_PPG_bufferHead++;
        }
    }
    else
```

- If the buffer head reaches the limit, the data is sent over the UART. As for MAX30102 buffer, an overlapping window is used between two different readings.

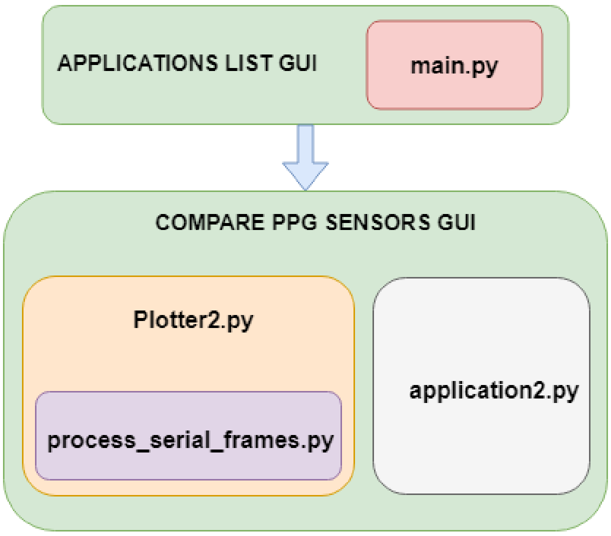
```c
    else
    {
        bBufferProcessed = FALSE;
        for (uint16_t i = CUSTOM_PPG_BUFFER_LENGTH - CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES; i < CUSTOM_PPG_BUFFER_LENGTH; i++)
        {

            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];
            CUSTOM_PPG_bufferGreen[(i - (CUSTOM_PPG_BUFFER_LENGTH - CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES))] = CUSTOM_PPG_bufferGreen[i];
        }
        /* Prepare serial frame structure */
        memset((void *) &frameParamCustom, 0x00, sizeof(frameParamCustom));
        frameParamCustom.frameType = CHANNEL_DATA;
        frameParamCustom.sensor = CUSTOM_SENSOR;
        frameParamCustom.params.wavelength = activeChannel;
        frameParamCustom.tissueDetected = TRUE;
        switch(activeChannel)
        {
            case IR_CHANNEL:
            pFrame = createSerialFrame((uint16_t *) &CUSTOM_PPG_bufferIR[CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES],
                                        (CUSTOM_PPG_BUFFER_LENGTH - CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES) * 2,
                                        (frameParams_t*) &frameParamCustom);
            break;
            case RED_CHANNEL:
            pFrame = createSerialFrame((uint16_t *) &CUSTOM_PPG_bufferRed[CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES],
                                         (CUSTOM_PPG_BUFFER_LENGTH - CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES) * 2,
                                         (frameParams_t*) &frameParamCustom);
            break;
            case GREEN_CHANNEL:
            pFrame = createSerialFrame((uint16_t *) &CUSTOM_PPG_bufferGreen[CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES],
                                        (CUSTOM_PPG_BUFFER_LENGTH - CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES) * 2,
                                        (frameParams_t*) &frameParamCustom);
            break;
        }
        sendFrame((uint8_t *) pFrame);
        bBufferProcessed = TRUE;
        CUSTOM_PPG_bufferHead = CUSTOM_NUMBER_OF_OVERLAPPING_SAMPLES;
    }
```

- Now that both buffers are sent over UART, build the project and program the target.



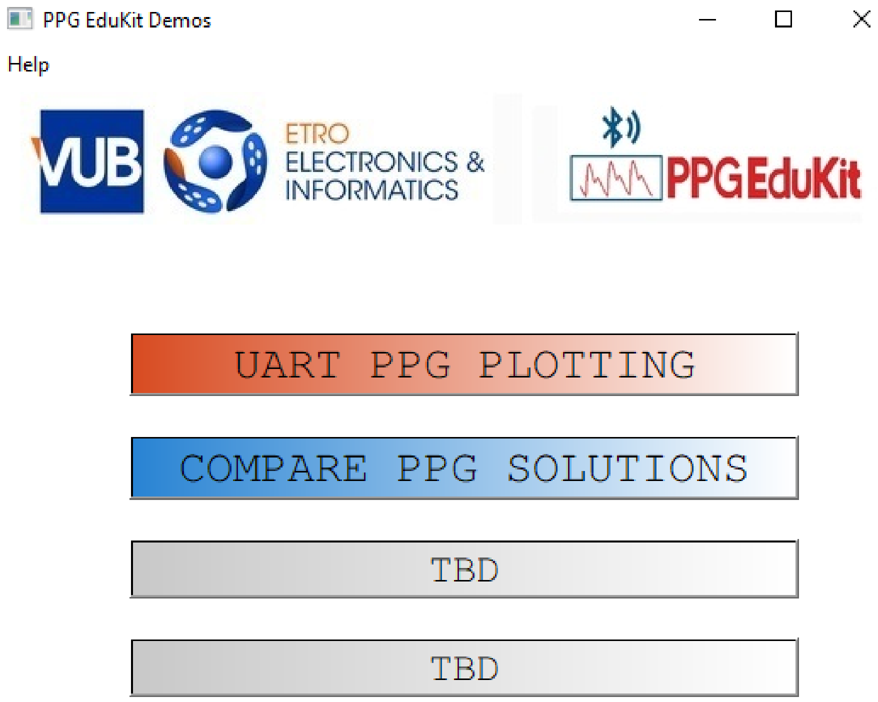
4 Run PPG EduKit GUI application

The PPG EduKit GUI is a Python program used to interface the PPG EduKit platform with different types of applications. The user interface is created using QT Designer and stored in **app2_gui.ui** file. The main window loads the requested application in the list. The application has two source files: **application2.py** and **Plotter2.py**. The **process_serial_frames.py** file is used to interface the embedded application that communicates over UART with the PC. If the data frame structure is changed (by adding more functionalities), the change should be included both in the **serial_frame.c** file and in **process_serial_frames.py** script.



To run the application, one can use just a simple terminal, but the recommendation is to use Spyder IDE or PyCharm.

- Open the command line and go to the location of the PPG EduKit GUI application.
- Execute the command: **pip install -r requirements.txt**. If the pip package is not installed, please refer to [5] in order to set up your environment.
- Open the project in Spyder/PyCharm.
- Run the **main.py** file. In the application list interface, select the **COMPARE PPG SOLUTIONS** application.



- Select the COM port and click **Connect**. Cover each sensor's surface and visualize the volumetric changes in arterial blood measured by two different sensors! Change the wavelength and the current to compare the optimal operating conditions for both modules. Also, one can change the sampling rate of both sensors in code and visualize the signal at different sampling rates.

```
void setup() {

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

  delay(1000);
  PPG_Shield.begin(&periphList, 70);
  delay(1000);

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


  PPG_Shield.enableLed(IR_CHANNEL, 5, true);

  curSwitch[AMP] = digitalRead(OLED_BUTTON_A);
  curSwitch[HPF] = digitalRead(OLED_BUTTON_B);
  curSwitch[LPF] = digitalRead(OLED_BUTTON_C);

  activeChannel = AMP;
  displayFilterText(AMP);
}
```

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