Freescale Semiconductor
Application Note

Document Number:AN4327
Rev. 2, 11/2012

Pulse Oximeter Fundamentals and Design

by: Santiago Lopez

Contents

1 Introduction

This application note demonstrates implementation of a basic pulse oximeter using Freescale products. The pulse oximeter is implemented using the Freescale medical-oriented, microcontroller Kinetis K53. This MCU embeds a 32-bit ARM®Cortex™-M4 processor, Ethernet, USB connectivity, and an analog measurement engine, ideal for medical applications.

This document is intended to be used by biomedical engineers, medical equipment developers, or any person related to the medical practice and interested in understanding the operation of a pulse oximeter.

2 Pulse oximetry fundamentals

This section contains information on the pulse oximeter function principle and basic physiology information about blood oxygenation.

2.1 Blood oxygenation

Body cells need oxygen to perform aerobic respiration. Respiration is one of the key ways a cell gains useful energy. The energy released in respiration is used to synthesize the adenosine triphosphate (ATP) to be stored. The energy stored in ATP can then be used to drive processes requiring energy, including biosynthesis, locomotion, or transportation of molecules across cell membranes.

Oxygen transportation is performed through the circulatory system. Deoxygenated blood enters the heart where it is pumped to the lungs to be oxygenated. In the oxygenation process, blood passes through the pulmonary alveoli where gas exchange (diffusion) occurs (Figure 1). Carbon dioxide (CO_2) is released and the blood is oxygenated, afterwards the blood is pumped back to the aorta.

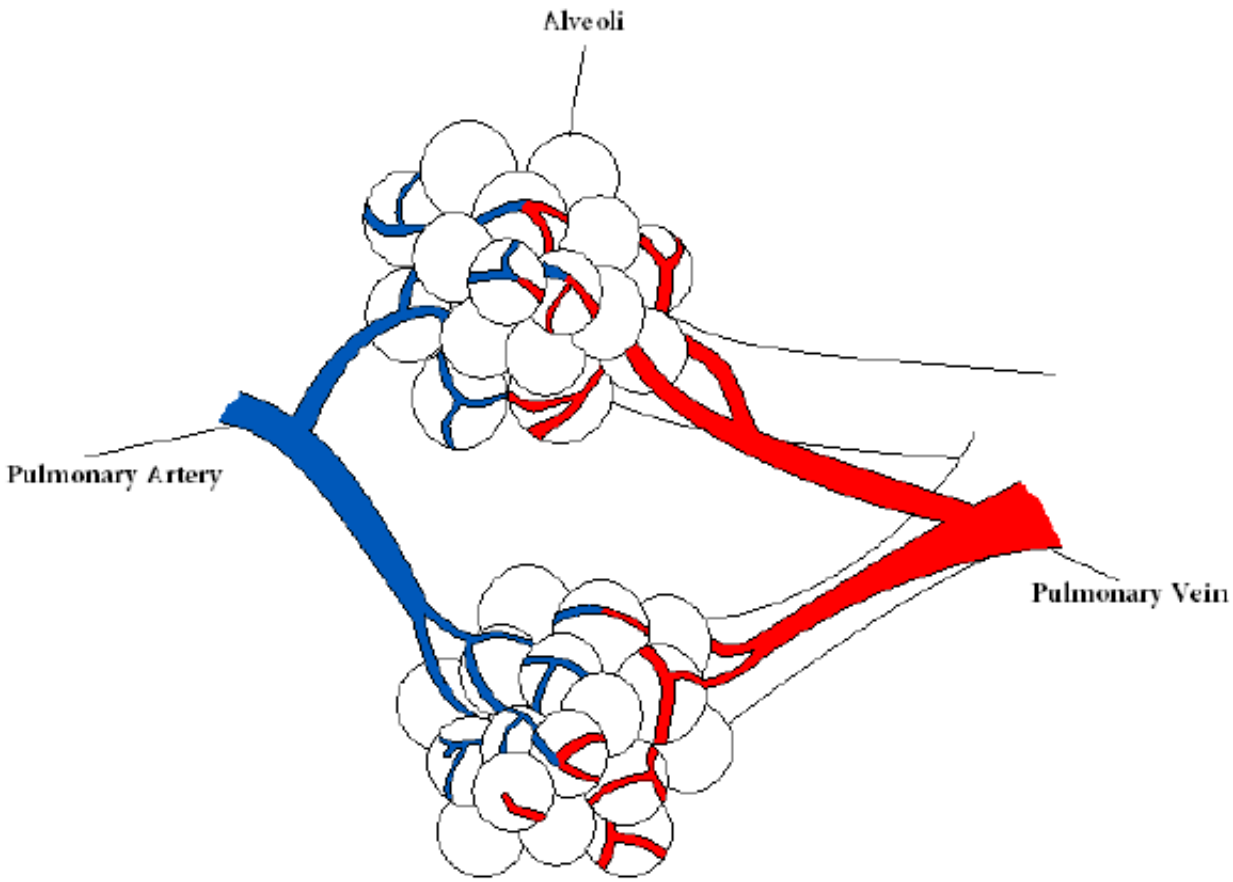

Figure 1. Pulmonary alveoli

Blood red cells contain a protein called hemoglobin. When oxygen reacts with this protein, it gets attached to it and generates Oxyhemoglobin (HbO_2). Red cells with oxygenated hemoglobin circulate in the blood through the whole body, irrigating tissues. When blood gets in contact with a cell, the red cell's hemoglobin releases oxygen and becomes Deoxyhemoglobin (Hb) (deoxygenated hemoglobin). At this point, blood without oxygen returns to the heart’s right atrium to repeat the process. The diagram below demonstrates the whole process (Figure 2).

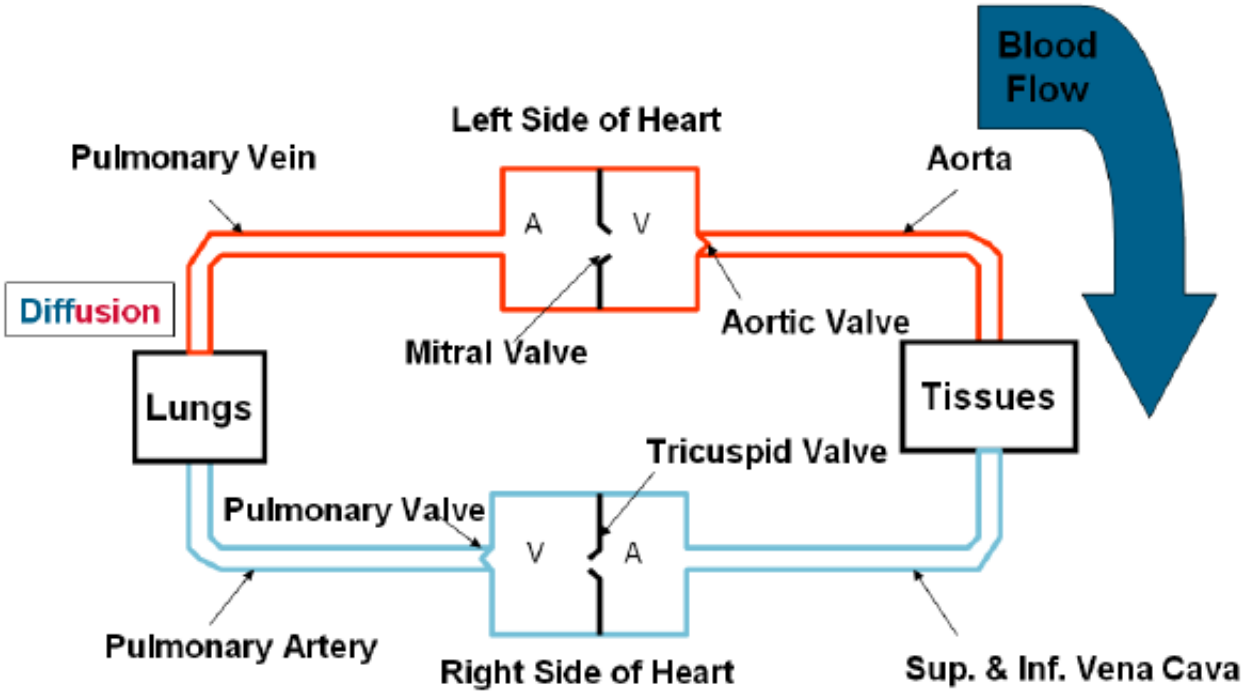

Figure 2. Blood circulation diagram

2.2 Pulse oximetry

Pulse oximetry is the non-invasive measurement of the oxygen saturation (SpO_2). Oxygen saturation is defined as the measurement of the amount of oxygen dissolved in blood, based on the detection of Hemoglobin and Deoxyhemoglobin. Two different light wavelengths are used to measure the actual difference in the absorption spectra of HbO_2 and Hb. The bloodstream is affected by the concentration of HbO_2 and Hb, and their absorption coefficients are measured using two wavelengths 660 nm (red light spectra) and 940 nm (infrared light spectra). Deoxygenated and oxygenated hemoglobin absorb different wavelengths. Deoxygenated hemoglobin (Hb) has a higher absorption at 660 nm and oxygenated hemoglobin (HbO_2) has a higher absorption at 940 nm (Figure 3).

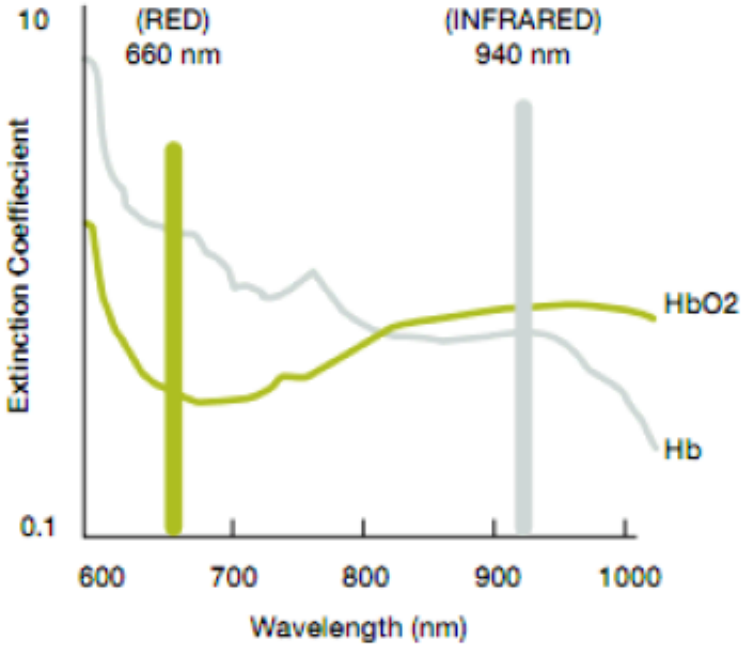

Figure 3. Hemoglobin light absorption graph

A photodetector in the sensor perceives the non-absorbed light from the LEDs. This signal is inverted using an inverting operational amplifier (OpAmp) and the result is a signal like the one in Figure 4. This signal represents the light that has been absorbed by the finger and is divided in a DC component and an AC component. The DC component represents the light absorption of the tissue, venous blood, and non-pulsatile arterial blood. The AC component represents the pulsatile arterial blood.

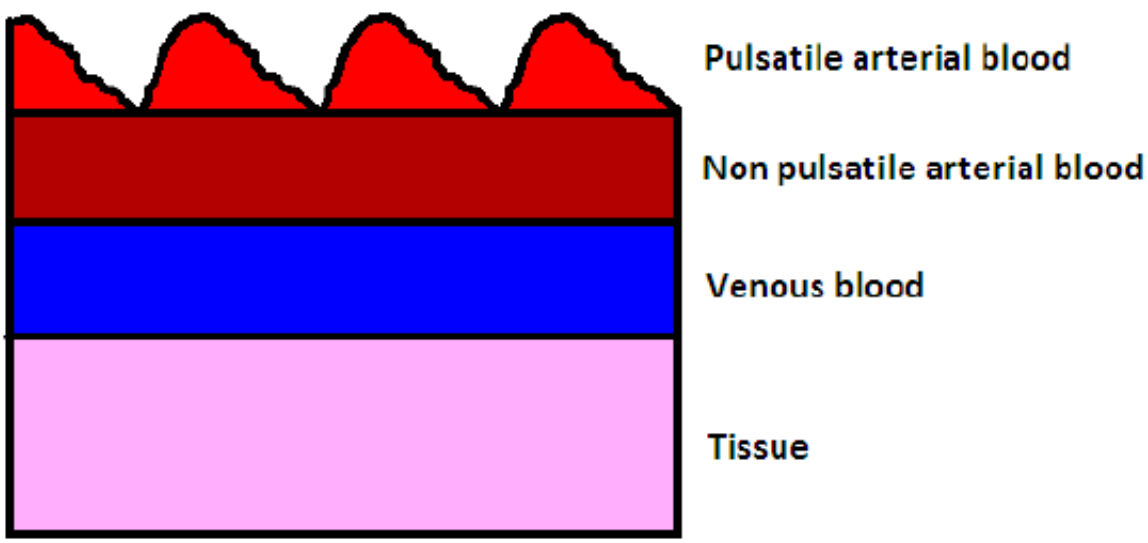

Figure 4. Light absorption diagram

The pulse oximeter analyzes the light absorption of two wavelengths from the pulsatile-added volume of oxygenated arterial blood (AC/DC) and calculates the absorption ratio using the following equation.

Equation 1:

$$\frac{(AC_{660})/(DC_{660})}{(AC_{940})/(DC_{940})}$$

SpO_2 is taken out from a table stored on the memory calculated with empirical formulas. A ratio of 1 represents a SpO_2 of 85 %, a ratio of 0.4 represents SpO_2 of 100 %, and a ratio of 3.4 represents SpO_2 of 0 %. For more reliability, the table must be based on experimental measurements of healthy patients.

Another way for calculating SpO_2 is taking the AC component of only the signal and determinate ratio by using following equation. SpO_2 is the value of RX100.

Equation 2:

$$R = \frac{\log_{10}(I_{ac})\lambda 1}{\log_{10}(I_{ac})\lambda 2}$$

I_{ac} = Light intensity at 1 (660 nm) or 2 (940 nm), where only the AC level is present.

A typical pulse oximetry signal is represented in Figure 5. The signal represents the pulsatile arterial blood absorption. The beats per minute can be calculated using this signal.

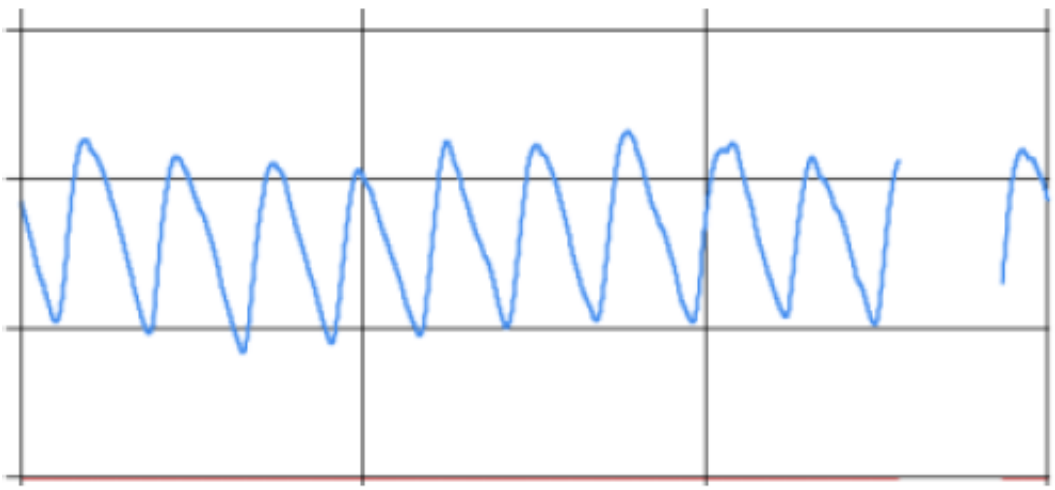

Figure 5. Typical pulse oximetry signal

3 Pulse oximeter implementation

The pulse oximeter is implemented using the Freescale MCU Kinetis K53 which embeds the following key features for the pulse oximetry signal treatment, among other medical oriented applications:

- 32-bit ARM® Cortex™-M4 core up to 100 MHz, bus speed up to 50 MHz
- DSP instructions for signal filtering
- Two Operational Amplifiers (OpAmp)
- Two Transimpedance Amplifiers (TRIAMP)
- USB connectivity as host, device or On-The-Go (OTG)
- Up to four pairs of differential and 24 single-ended 16-bit ADC channels
- 3 x 16-bit Flex Timer Modules (FTM) with PWM capability

The Kinetis K53 integrates most of the peripherals needed for pulse oximeter implementation, although some external components are required. These components are integrated in an external Analog Front End, described below.

3.1 MED-SPO2 analog front end

Freescale Analog Front Ends (AFEs) provide fast prototyping capabilities enabling medical equipment developers to reduce time to market. MED-SPO2 AFE includes all the necessary external components (except sensor) to implement a pulse oximeter together with the Kinetis K53 MCU. The AFE functional block diagram is shown and described below (Figure 6).

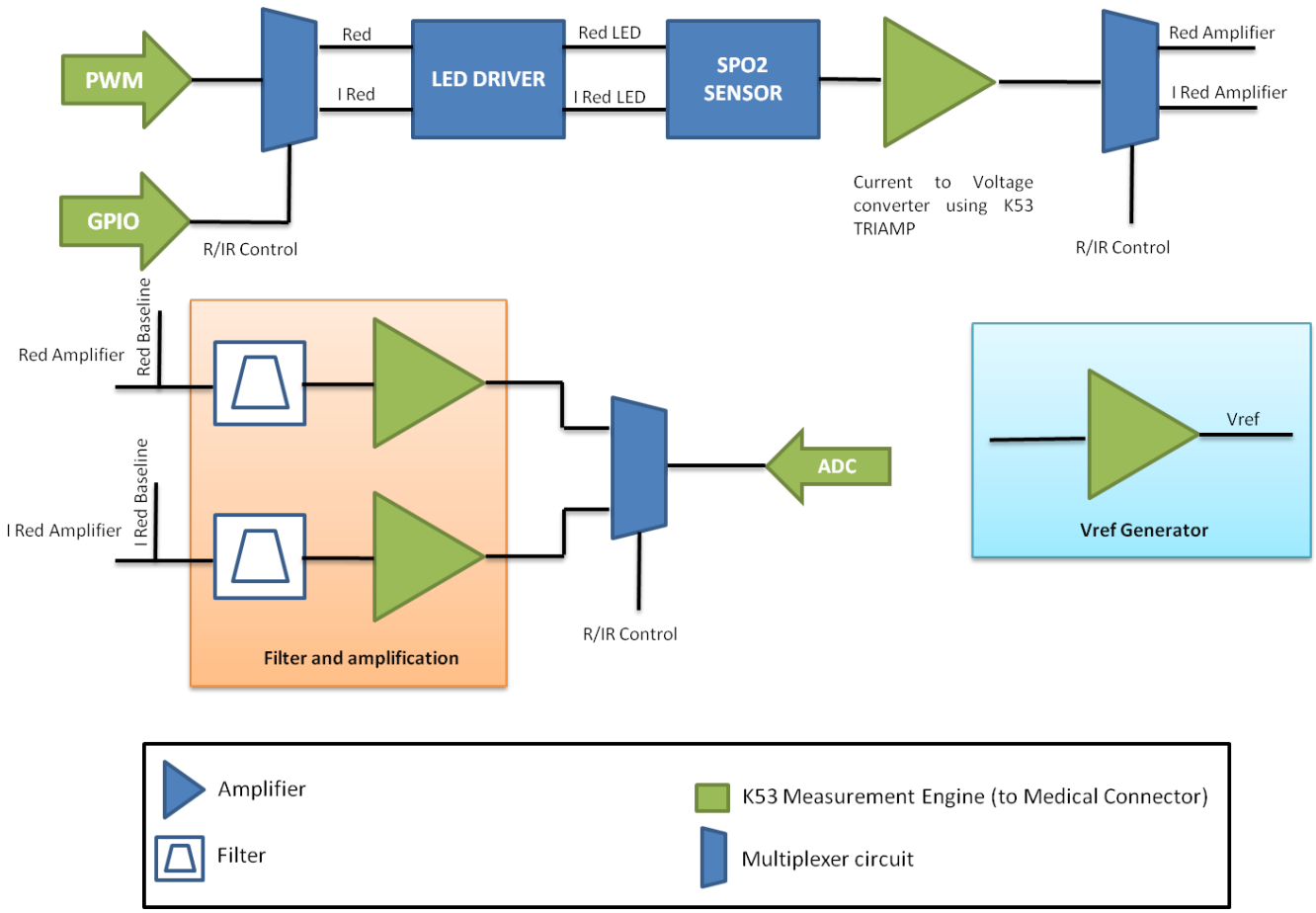

Figure 6. MED-SPO2 functional block diagram

3.1.1 Medical connector

The medical connector is a standard connector in Freescale medical-oriented boards (TWR-9S08MM, TWR-MCF51MM and TWR-K53). This connector includes the most important analog peripherals for medical applications and an I^2C channel for communication. The following table describes medical connector signals.

Table 1. Medical connector signals

1	VCC (3.3V)	VSS (GND)	2
3	I2C SDA	I2C SCL / PWM	4
5	ADC Differential CH +	ADC Differential CH -	6
7	ADC Single Ended CH	DAC Out	8
9	Op-Amp 1 Out	Op-Amp 2 Out	10
11	Op-Amp 1 Input -	Op-Amp 2 Input -	12

Table continues on the next page...

Table 1. Medical connector signals (continued)

13	Op-Amp 1 Input +	Op-Amp 2 Input +	14
15	TRIAMP 1 Input +	TRIAMP 2 Input +	16
17	TRIAMP 1 Input -	TRIAMP 2 Input -	18
19	TRIAMP 1 Out	TRIAMP 2 Out	20

3.1.2 Multiplexer circuit and LED driver circuit

The pulse oximeter needs two different wavelengths to perform measurements. These wavelengths are generated using two Light Emitter Diodes (LEDs), a Red LED (660 nm,) and an Infra Red LED (940 nm). Samples cannot be taken at the same time because there is only one photodetector for two signals, therefore signals must be multiplexed. MED-SPO2 includes a GPIO-controlled analog multiplexer that allows selecting the wavelength to be sampled.

LED intensity is controlled using a PWM signal. However, MCU PWM pins do not provide enough strength to drive LEDs in a proper manner. A LED driver circuit provides the LED with sufficient energy to work. Figure 7 shows a basic LED driving circuit.

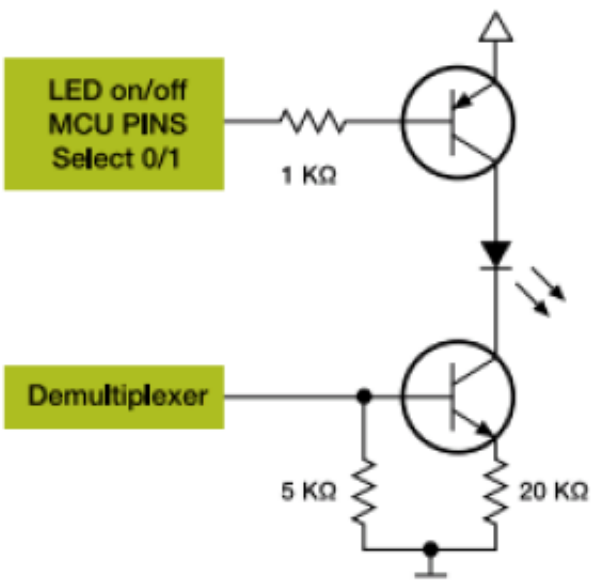

Figure 7. LED drive circuit

3.1.3 Pulse oximeter sensor

The MED-SPO2 was designed for working with Nelcor-DS100 series sensors or any other that are compatible. The sensor is connected to the board through a simple DB9 connector, similar to the one used in the RS232 communications standard. The following image (Figure 8) shows connections with the sensor using a DB9 connector.

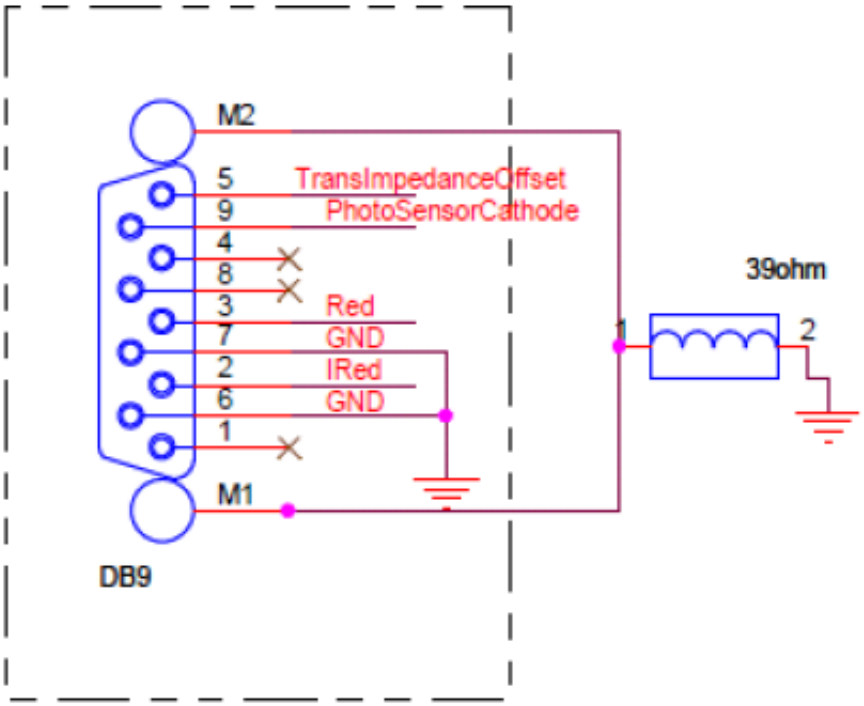

Figure 8. DB9 Board connections to sensor

The pulse oximeter sensor already contains both LEDs, Red, IRed, and the photodetector needed for light absorption detection.

3.1.4 Current to voltage converter

The output generated by the photodetector is a current that represents the light absorption. This current needs to be converted into a voltage in order to be properly filtered and treated. Conversion is performed using a current to voltage converter which consists in a single supply, low input offset voltage, low input offset and bias current TRIAMP embedded on K53 together with a feedback resistance and a capacitor for filtering purposes. Figure 9 shows the implemented circuit.

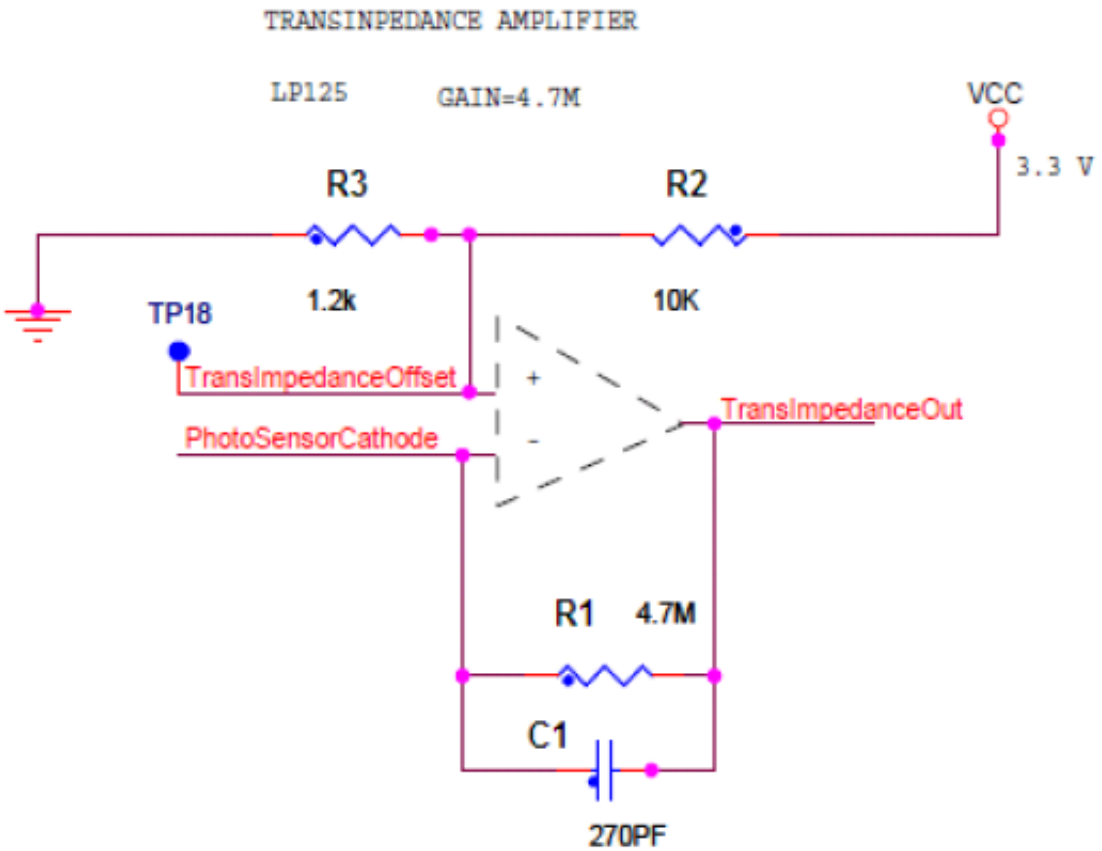

Figure 9. Current to voltage converter

This circuit combines a current to voltage converter and a low-pass filter to improve the signal treatment. The output voltage and cut frequency are given by the following formulas:

Equation 3:

$$V_{out} = \text{PhotoSensorCathodeCurrent} * R1$$

Equation 4:

$$\text{FC} = \frac{1}{2\pi(R1*C1)} = 125\,\text{Hz}$$

A 125 Hz low-pass filter is to remove high frequency noise from the received signal.

3.1.5 Filtered and amplification

This block is divided in five filters, four of them are passive filters and one of them is an active filter. Both 660 nm and 940 nm signals are processed using these filters for noise elimination and amplification. Figure 10 shows the filter circuit.

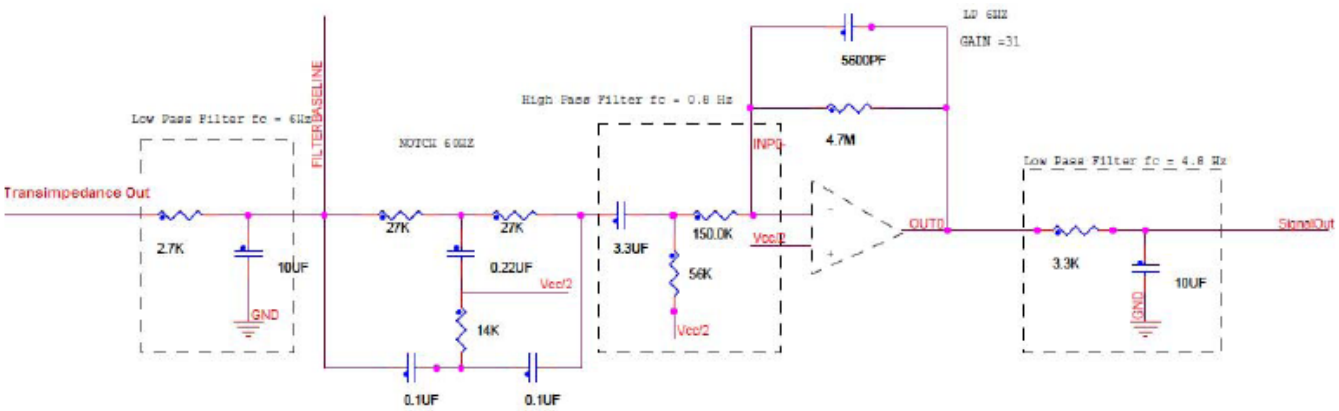

Figure 10. Filter circuit

The first filter is a low-pass filter with a cutoff frequency of 6 Hz designed to eliminate high frequency noise. The following equation obtains the filter cut frequency:

Equation 5:

$$f_0 = \frac{1}{2\pi RC}$$

The second filter is a 60 Hz notch filter. The purpose of this filter is to eliminate the 60 Hz line interference. Figure 11 represents the connections for a notch filter.

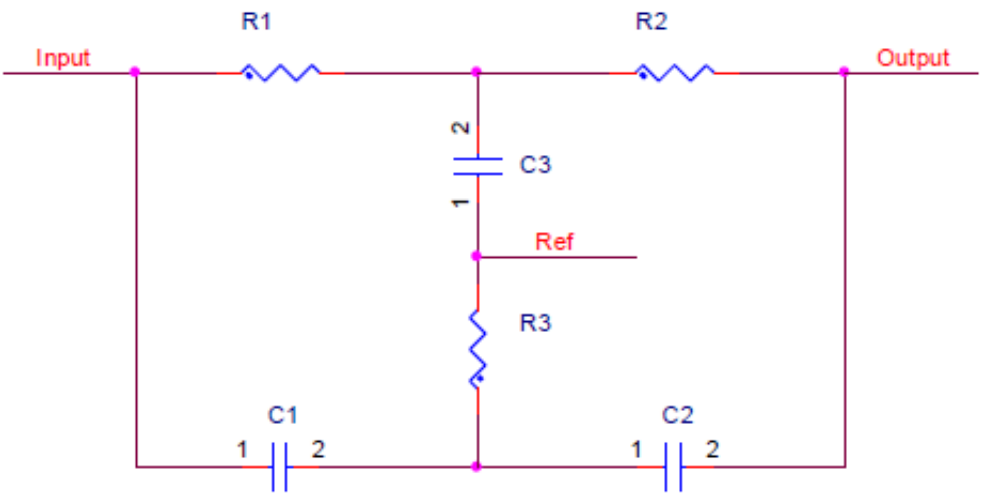

Figure 11. Notch filter

The notch filter is referenced to VCC/2 to add an offset voltage. The notch filter cutoff frequency and design parameters are represented by the following equations.

Equation 6:

$$f_0 = \frac{1}{2\pi R1C1}$$

Equation 7:

$$R1 = R2 = 2R3$$

Equation 8:

$$C1 = C2 = \frac{C3}{2}$$

The third filter is a 0.8 Hz high pass filter. The cutoff frequency of this filter is calculated using Equation 5. This filter removes the DC component of the signal.

The fourth filter is a first order active 6 Hz low-pass filter that also provides a gain of 31. Figure 12 shows an active low pass filter.

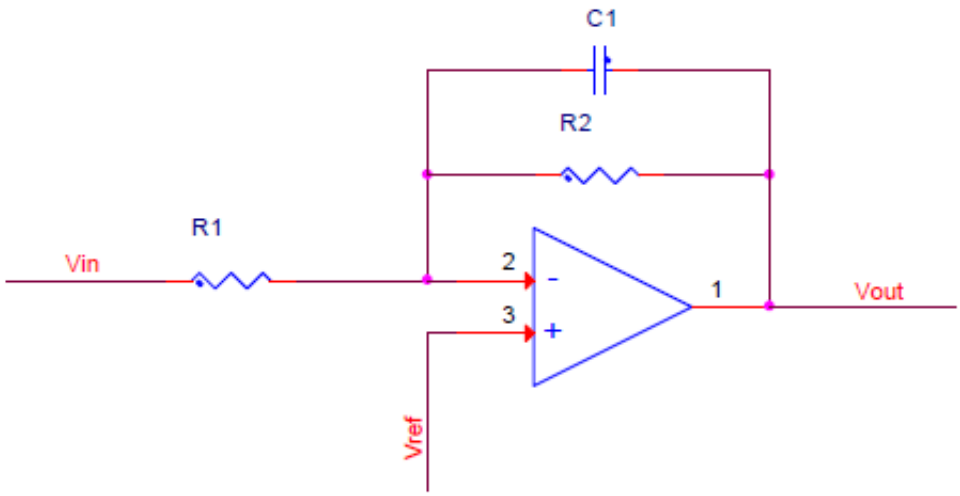

Figure 12. Active low-pass filter

This filter uses a K53 internal OpAmp to be developed. Vref on positive input allows using an inverter amplifier. The following equations determine the filter cutoff frequency and gain.

Equation 9:

$$f_0 = \frac{1}{2\pi R2C1}$$

Equation 10:

$$A = -\frac{R2}{R1}$$

The last one is a 4.8 Hz low pass filter similar to the first one (Equation 5), and the equation that represents this filter is the same that represents the first one.

3.1.6 Vref generator

Because a negative voltage source implies extra costs and components, analog signals are handled in a positive voltage range. Nevertheless some analog signals like some biopotentials have negative voltages that need to be considered. This is solved mounting the AC signal on a positive DC level, typically VCC/ 2. Vref generator produces this signal using a simple voltage divisor and a K53 embedded OpAmp. Figure 13 shows a Vref generator circuit.

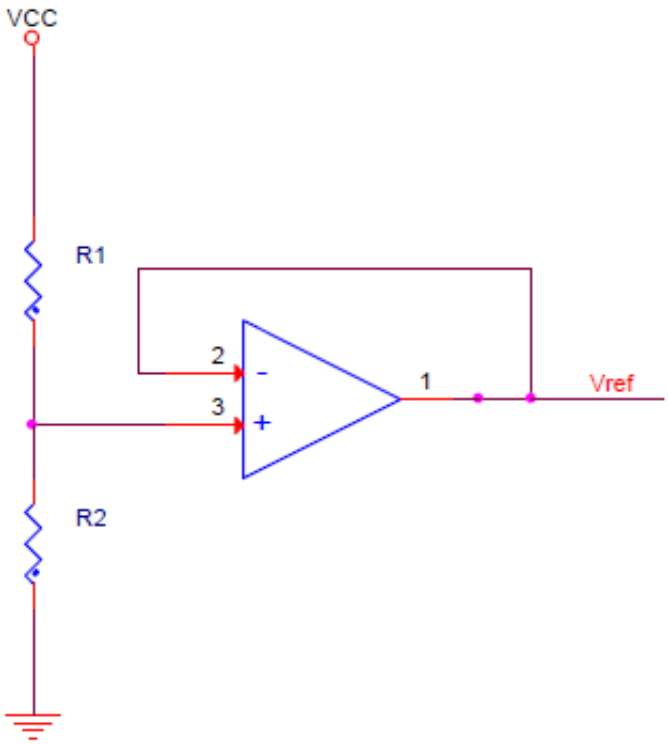

Figure 13. Vref generator circuit

3.2 Functional description

The measurement starts when the MCU generates a PWM signal that varies the LED intensity. Because every person has a different finger size and skin color, the LED needs to be calibrated to acquire an accurate signal. LED calibration is performed by taking the LED filtered baseline (see Figure 10) and using an algorithm described in the Software model which changes the PWM duty cycle value to adjust the LED intensity for every kind of user.

The LED driver circuit helps to drive LEDs so that power is not provided directly by the MCU. Using transistors, the LEDs are powered directly by the VCC line and controlled by the MCU. The switch control pin on the MCU selects which LED is turned on at that time. Light from Red and IRed LEDs on the sensor travels through the finger and the non-absorbed light is received in the photodetector (Figure 14).

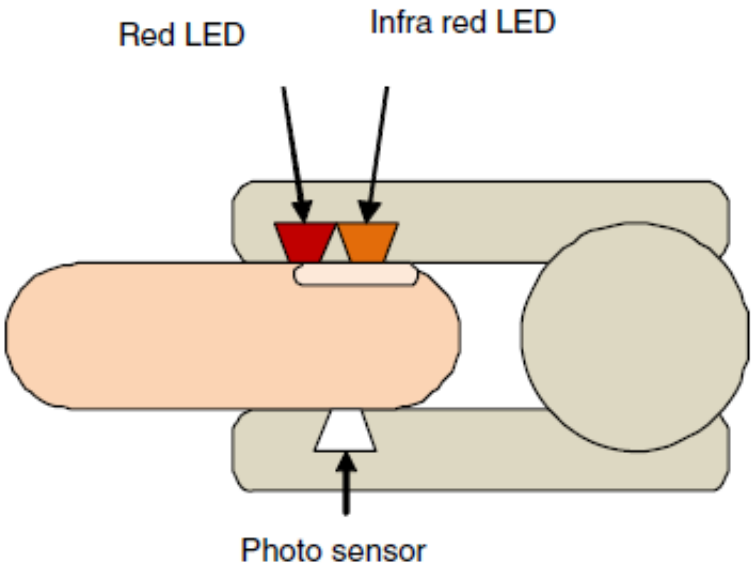

Figure 14. Pulse oximeter sensor

The signal passes through a current to the voltage converter (Figure 9) where it is filtered, amplified, and converted into a voltage. The signal is now multiplexed to its respective filter and amplification stage, depending on whether it is Red or IRed LED. At this stage, the signal is treated and most of the noise is removed. The signal is also amplified in order to be detected easily by the MCU ADC. The filtered signal is then sent to an ADC channel on the MCU.

One sample of the Red filtered signal, Red baseline, IRed filtered signal, and IRed baseline are taken every 1 ms. Samples are captured using the embedded 16-bit ADCs and filtered using a 0.5 Hz to 150 Hz FIR (Finite Impulse Response) software filter on the Kinetis K53 MCU for high frequency and DC component removal, taking advantage of the MAC (Multiply and Accumulate) DSP instruction. Samples are stored on a software buffer and averaged. A peak detection algorithm is used to determinate the AC component of the signal that is generated by the pulsatile arterial blood absorption. This is the part of the signal which is used for SpO_2 and beats per minute (bpm) calculation. The samples taken and the calculated data (SpO_2 and bpm) are sent to a GUI on a computer. More information about the software process can be found in Software model.

4 Software model

The pulse oximeter demo is based on the Freescale USB stack and behaves as a USB CDC (Communication Device Class). The demo works using state machines that execute one state per cycle avoiding CPU kidnapping and emulating parallelism.

Figure 15 shows the general software model.

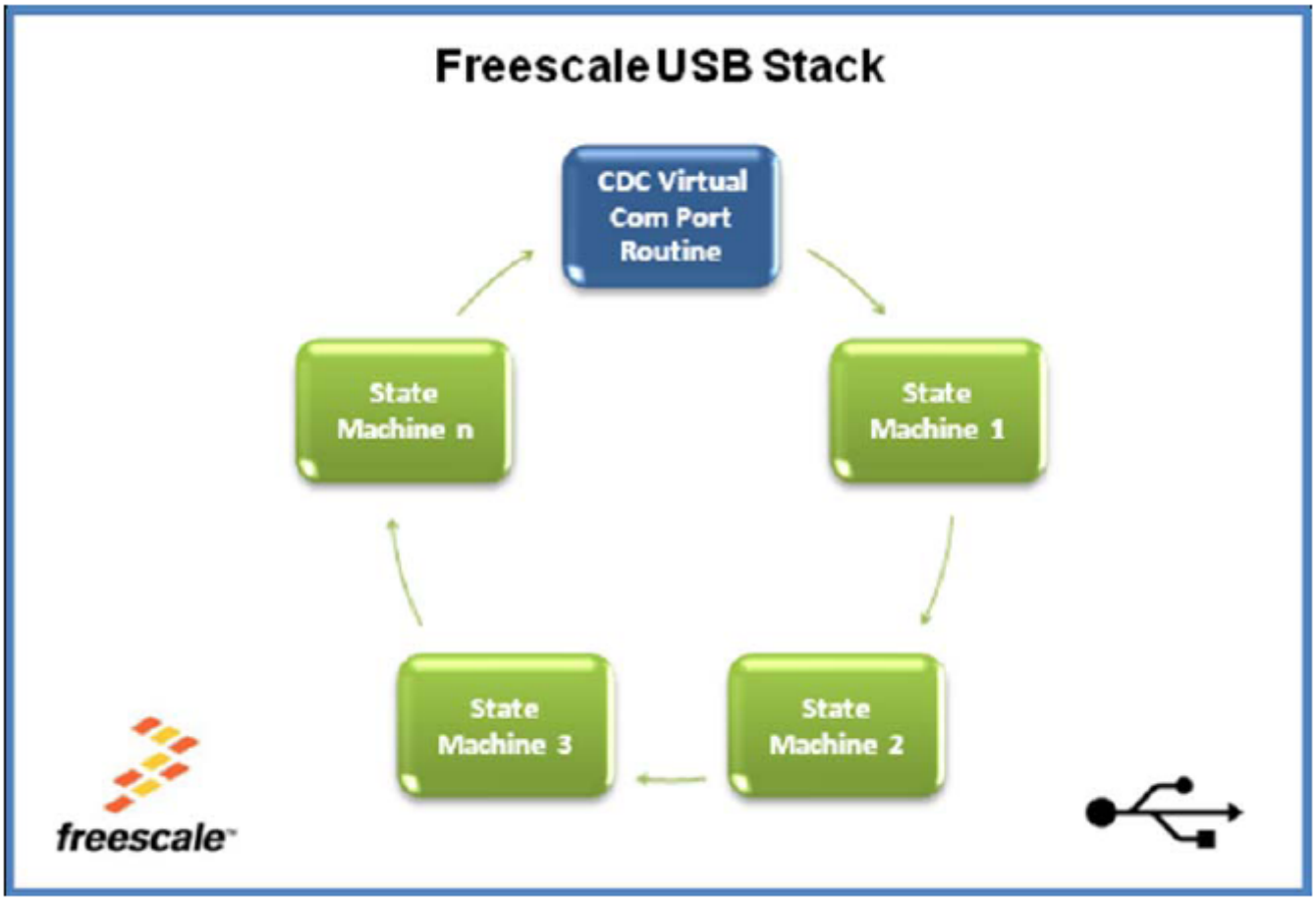

Figure 15. General software model

Every state machine is a task that has to be performed by the MCU. The system can perform several tasks completing one at time and running the next one until the first one is accomplished in a FIFO (First In First Out) organization. Some state machines contain sub-state machines allowing the MCU to distribute the CPU load equally. As mentioned before, software is based on the Freescale USB Stack with PHDC. More information about this software can be found in USBAPIRM: Freescale USB Stack with PHDC Device API Reference Manual, available at **freescale.com.**

MED-SPO2 software is basically divided in three main parts:

- Initialization
- Communication with PC
- Measurement Execution

4.1 Initialization

The file main_kinetis.c contains the microcontroller initialization routines and calls the main application routine. The main application routine (Figure 15) contains all the other state machines and calls them in a FIFO order. Initialization is integrated by several functions described below. The most relevant ones are Init_Sys which initializes all the peripherals required by the MCU for USB initialization and TestApp_Init which initializes functions required by the application itself.

4.1.1 Init_Sys()

Init_Sys() initializes the basic functions of the microcontroller, like clock initialization. First, this function copies the vector table to RAM and clears any pending interrupts on USB and then, it enables interrupts from the USB module.

At this point, pll_init() is called. This function configures the microcontroller for running in PLL mode at 96 MHz of the core speed with a 50 MHz clock generating a system bus frequency of 48 MHz. This is important because the USB module needs 48 MHz to run properly. More information about changing among clock modes can be found in K53P144M100SF2RM: K53 Sub-Family Reference Manual for 100 MHz devices in 144 pin packages, availabe on **freescale.com**.

After PLL initialization, the MPU_CESR is cleared and clock gating for USB is configured to get 48 MHz from the system bus.

4.1.2 TestApp_Init();

The TestApp_Init()—Contains all the initializations needed by the MED-SPO2 application and for USB configuration as a CDC device. This function is contained in the file Main_App.c and is called only once at the beginning of the program.

GPIOS_CLOCK_INIT—Is a macro that enables all the GPIO clocks for future use. Kinetis disables all its peripheral clocks to save power, and the peripherals must be activated enabling its respective bit on the respective SIM_SCGCx register.

vfnEnable_AFE()—Initializes GPIO settings for the pin used as an AFE enabler which polarizes a field effect transistor (FET) which is used as a switch for power on and off in the AFE on the medical connector.

SwTimer_Init()—Initializes the software timer. The software timer is a function that allows executing a function when a programmed period of time has elapsed. More information about these routines can be found in Software timer.

After SwTimer_Init is executed, the Kinetis K53 USB module is initialized as a CDC device calling the function USB_Class_CDC_Init. At this point, USB is now functional and can be recognized by a PC. Information about USB_Class_CDC_Init and other USB Stack related questions can be found in USBAPIRM: Freescale USB Stack with PHDC Device API Reference Manual, available at **freescale.com**.

4.2 Communication with PC

After the MCU has been initialized, TestApp_Task is executed in an infinite loop. This function contains the main state machine and executes the communication and measurements state machines (Figure 16).

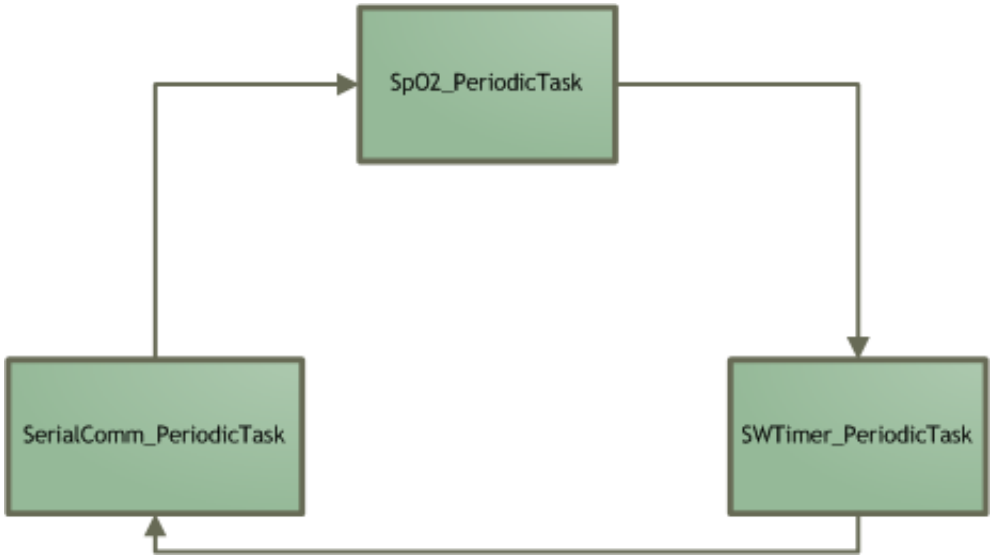

Figure 16. TestApp_Task diagram

SpO2_PeriodicTask executes all the sub-states related to the pulse oximetry measurement. This function is later explained in detail in the following sections. SWTimer_PeriodicTask checks if the programmed times for running programs has elapsed. For more information about this routine, see Software timer.

4.2.1 SerialComm_PeriodicTask

This function checks if there is new data from the PC. When data exists in the USB buffer this function analyzes it and executes a routine when it is a valid command. Figure 17 represents the SerialComm_PeriodicTask function.

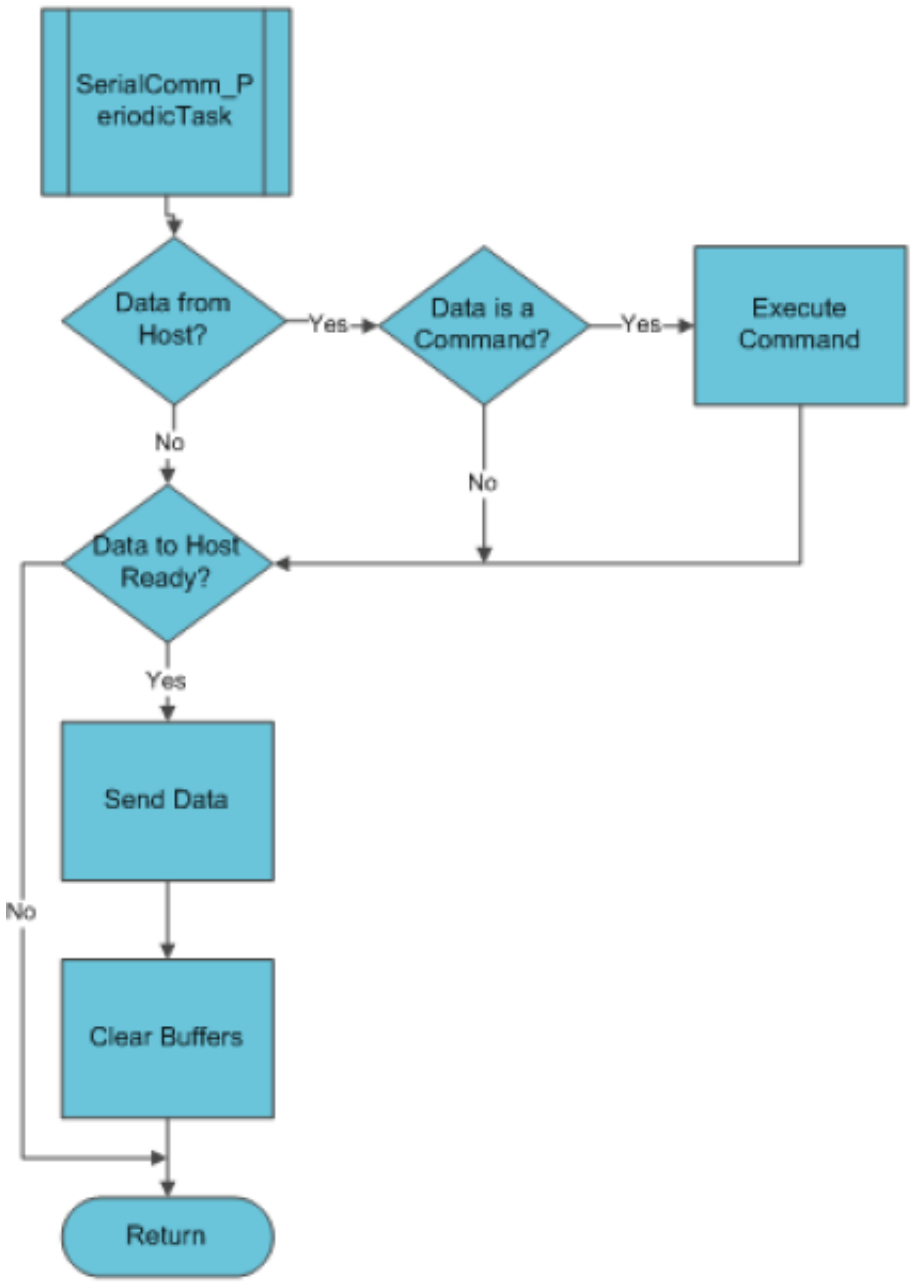

Figure 17. SerialComm_PeriodicTask flow diagram

Communication routines are based on the Freescale USB with PHDC USB Stack Communication Device Class functions. The SerialComm_PeriodicTask function is constantly checking the g_recv_size flag. When this flag has a value other than 0, this means that there is some data received from a computer on the USB buffer. This information is copied to the main application InBuffer and the received data quantity is copied to InSize. After that, g_recv_size is set to 0 indicating that data has been read.

When InSize has a value other than 0, it is understood that data has been received, and it is checked to be compatible with the communication protocol data frame (Figure 18).

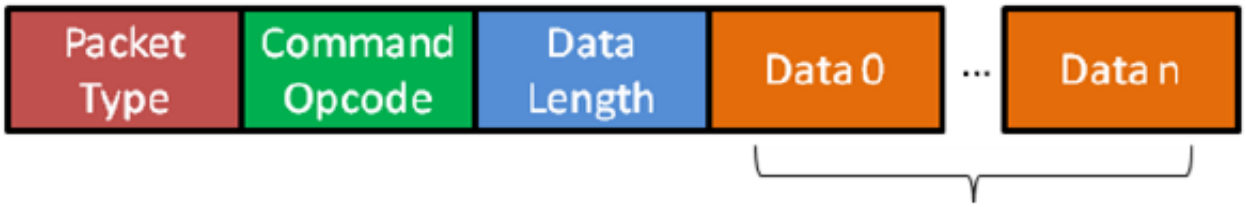

Figure 18. Protocol data frame

The first data byte checked is the Packet Type, this byte defines the kind of packet that has been received. The application only recognizes Request (REQ) packets. Confirmation Packets (CFM) and Indication Packets (IND) are recognized only by the PC GUI. For more information, see Communication protocol.

If the first byte received is a Request command, the second byte is checked to determine which command wants the PC to be executed. There are several kinds of Command Opcodes.

MED-SPO2 responds to only four:

- SpO2StartMeasurementReq
- SpO2AbortMeasurementReq
- SpO2DiagnosticModeStartMeasurementReq
- SpO2DiagnosticModeStopMeasurementReq

One of these commands is specified on the Command Opcode byte. The selected function is executed using the pointer to function ExecuteCommandReq[InBuffer[COMMAND_OPCODE]].

4.2.2 SpO2StartMeasurementReq and SpO2DiagnosticModeStartMeasurementReq

These functions are very similar; both of them send a confirmation packet and start pulse oximetry measurements. The only difference is that SpO2StartMeasurementReq initializes MED-SPO2 for a single measurement and SpO2DiagnosticModeStartMeasurementReq initializes MED-SPO2 for a continuous measurement. More details about this can be found in Measurement execution.

At the beginning, the function vfnStartSpO2MeasurementEngine initializes the OPAMPs and TRIAMPs for working in general-purpose mode.

SpO2_Init is called into the subroutine and initializes all the timers needed by the application. FTM1 is needed by the measurement routine to control the ON/OFF times of the sensor LEDs. FTM2 is needed for PWM generation. SpO2_INIT_SWITCH_CONTROL_PIN is also called to initialize the GPIO pin for multiplexing control.

After vfnStartSpO2MeasurementEngine is executed, the measurement start request function generates a confirmation packet indicating to the PC that a command has been received. Depending on the measurement start request function called, the respective start measurement function is called. In this example, because SpO2DiagnosticModeStartMeasurementReq has been called, the function calls SpO2_DiagnosticModeStartMeasurements as the start measurement function. This function returns TRUE or FALSE depending if the initialization was completed successfully or not. Depending on this, the function completes the confirmation packet with the last byte indicating if the request was executed correctly or not. The packet is then sent using the function SerialComm_SendData.

This function checks OutSize to be greater than zero. If this condition is true, it is assumed that there is information in the OutBuffer. This information is copied to the USB send buffer called g_curr_send_buf. Information in this buffer is sent to the PC and the OutSize variable is cleared.

4.2.3 SpO2AbortMeasurementReq and SpO2DiagnosticModeStopMeasurementReq

When a stop or abort measurement is requested by the PC (depending on the Command Opcode), one of these functions is executed.

Confirmation packets for stop/abort measurement requests do not require the last byte containing error information. It is always assumed that a stop/abort request has been executed when the confirmation packet is received. The function first sends a confirmation packet indicating to the PC that the request command has been received. After that, a command depending on the stop/abort request called, is executed. In this example because SpO2DiagnosticModeStopMeasurementReq has been called, the function SpO2_DiagnosticModeStopMeasurement is executed. After the measurements have stopped, the confirmation packet is sent using the function SerialComm_SendData. The function vfnTurnOffMeasurementEngine is now called; this function disables all the OPAMPs and TRIAMPs.

4.3 Measurement execution

There are three basic parts on measurement execution:

- Measurements Start
- Measurements Execution
- Measurements Stop

4.3.1 Measurements start

When a host has sent a SpO2 Start Measurements request to the appropriate function, according to the Command Opcode received, it is executed. Each of these functions calls a start measurement routine.

SpO2_DiagnosticModeStartMeasurement first checks if the SpO2 state machine is in an idle state, otherwise it means that the state machine has been previously initialized. If it is in an idle state, the function initializes ADCs needed for both, the main signal and baseline measurements. Then, depending on the function called, SpO2IsDiagnosticMode is set to true or false. If SpO2IsDiagnosticMode is true, the device will send continuous SPO2 measurements and graph data. If it is false, the device will take 10 samples and if those samples are valid the device averages those samples and sends a single SpO_2 measurement.

The SpO2 state machine's actual state is set to SPO2_STATE_MEASURING to indicate to the device that pulse oximetry measurements are being performed. The ResetVariables() function resets all the SPO2 variables to its original state and clears all the buffers. The SwTimer_StartTimer(TimerSendGraphData, SPO2_UPDATE_GRAPH_PERIOD) starts a software timer for a graph update. More information about this routine see Software timer.

4.3.2 Measurements execution

When a measurement start function has been called, the SPO2 state machine is set to SPO2_STATE_MEASURING indicating to the device that measurements are going to be performed. The SPO2 state machine calls the StateMeasuring function which is a sub-state machine containing the necessary steps to perform measurements. Figure 19 shows the Sub-State Machine flow diagram.

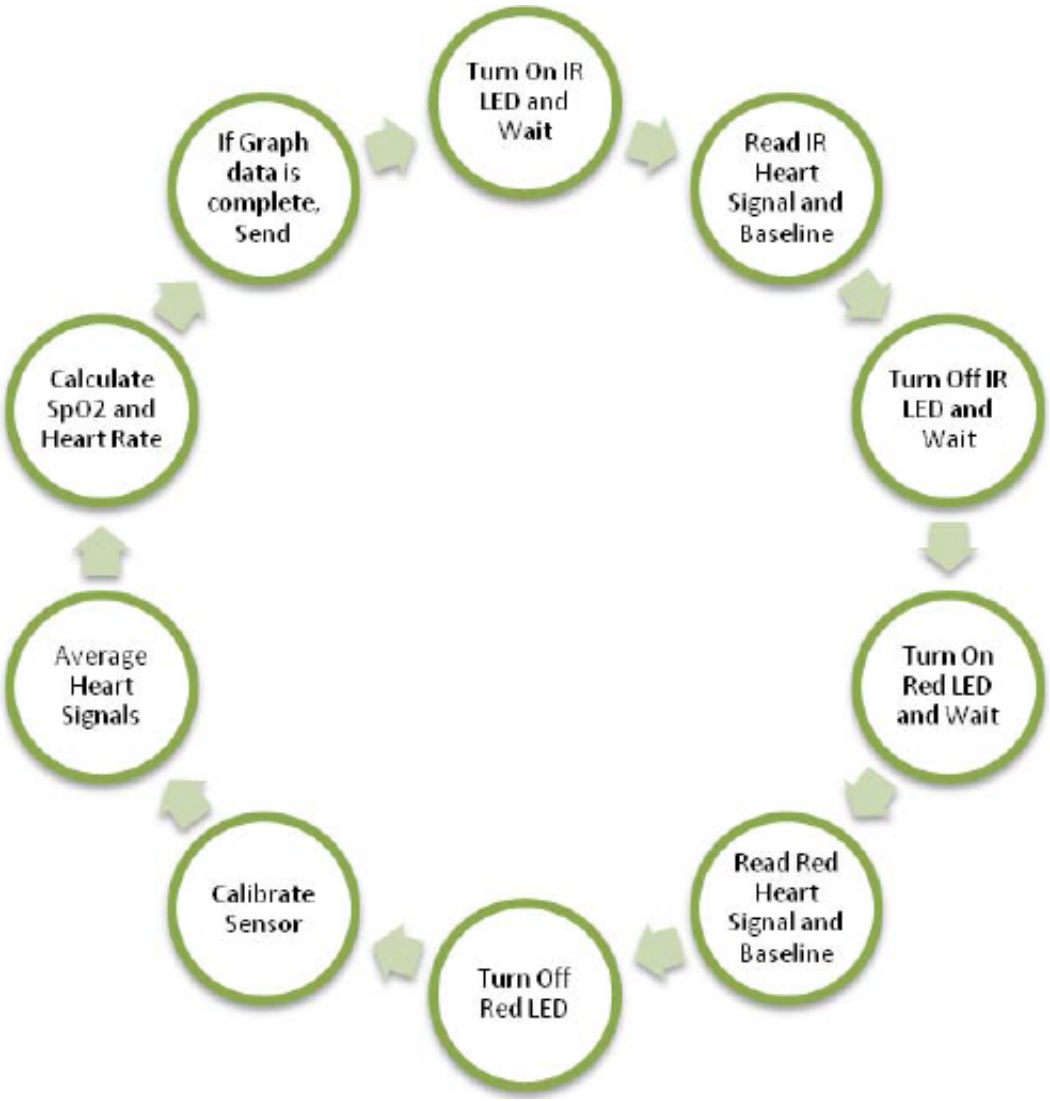

Figure 19. StateMeasuring flow diagram

StateMeasuring is a sub-state machine, this means that sub-states are called one by one every time the StateMeasuring function is called.

To take a sample, follow the steps below:

1. Turn on Infra Red (IR) LED
2. Wait signal to stabilize
3. Read IR signal and baseline
4. Turn off IR LED
5. Wait signal to stabilize
6. Turn on Red LED
7. Wait signal to stabilize
8. Read Red signal and baseline
9. Turn off Red LED

The first step is to turn on the IR LED. The function SPO2_SET_SWITCH_CONTROL changes the value of the pin that controls the board multiplexor and controls which signal is going to be read. The Pwm_Start(PwmValue) starts generating the PWM signal which turns on the selected LED and controls light intensity by changing the PWM duty cycle. The PwmValue determinates the duty cycle and therefore LED intensity. This value is modified in the LED calibration routine to adjust the LED intensity for working with different skin colors and finger widths.

The START_TIMER_us function starts the microseconds counter which is constantly being checked to determine when the predefined wait time has elapsed. This is performed to generate the signal stabilization times. Then the next sub-state is called.

When the wait time has elapsed, the time counter is set to 0 to generate a new wait time for the Red LED. The SpO2 signal and baseline are read from their respective ADC channels. The SPO2_FIR is a macro that determines if a FIR (Finite Impulse Response) filter is used or not. When defined, the samples taken are filtered using the functions in Kinetis_FIR.c which uses the MCU embedded DSP instructions to perform the filter function. Samples are then averaged and stored into a buffer using the function MovingAverage_PushNewValue16b to make calculations. This process is executed again, but this time taking samples from the Red LED signal.

After the samples have been taken, calculations are performed and LEDs are recalibrated. Case 4 on the state measuring function performs these actions.If Diagnostic Mode was initialized when the IsTimeToUpdateGraphData flag on TimerSendGraphData function is set, the data on the IR signal buffer is stored using the function SaveGraphData sent later to the PC, then the TimerSendGraphData starts again.

The LED intensity is recalibrated when TimeToRecalibrate is equal to zero, on the calibration routine the baseline signal average results are compared with baseline thresholds to determine if calibration is needed. If it is necessary to recalibrate, the respective Calibrate function is called.

For example, if IR baseline signal is out of limits, the CalibrateIRLedIntensity function is called. This function compares the baseline signal against several threshold values and changes the LED PWM duty cycle until the baseline signal is in the limits.

After calibration has been checked, function FindMaxAndMin is called. This function determines the maximum (Max) and minimum (Min) peaks of the signal in a window interval, that is, the function determines the biggest and the smallest sample in a determined period of time. Once the Max peak and Min peak have been determined, the DoCalculations function is called.

The DoCalculations function calculates the SpO_2 percentage and the heart rate. The function first determines the peak to peak voltage subtracting the Min peak to the Max peak to remove the DC component of the signal. The Vrms voltage is then calculated by using the formula:

$$Vrms = 0.5 * \frac{Vpp}{\sqrt{2}}$$

SpO_2 percentage is calculated in relation with the current received from the IR LED and the current received from the Red LED. Log_{10} of the RMS voltage for each of the different wavelength signals is calculated using a Look Up table containing the different logarithm results. Ratio is calculated using the $Log_{10}(I_{ac})$ formula being SpO_2 equal to RatioX100 (See Equation 2 in Pulse oximetry fundamentals).

The heart rate is calculated using the variable SpO2SamplesBetweenPulses. This variable contains the count of samples taken between two heart beats. Because one sample is taken every 1 µs, the heart rate is calculated by 60,000 divided by the quantity of samples between pulses.

SpO_2 and HR calculations are stored in their respective 10 bytes array which contains the last 5 measurements taken. Every array is averaged on the function CalculateHeartRateAndSpO_2 to avoid false measurements. The result of every average is the information sent to the PC.

4.3.3 Measurements end

When the host requests the device to stop SpO2 measurements, the function SpO2_AbortMeasurement is called. This function is also called in diagnostic mode.

This function places the SpO2 State Machine in Idle State to stop taking measurements. After this, the timer for sending graph data and the timer for generating the wait times are stopped. Finally, the PWM is turned off.

5 Running MED-SPO2 demo

MED-SPO2 demo was developed for running on the Freescale Tower System enabling fast prototyping and a short time to market. The following steps will guide the user to run the MED-SPO2 demo.

5.1 Tower system configuration

The MED-SPO2 demo needs the following components to be assembled (Figure 20).

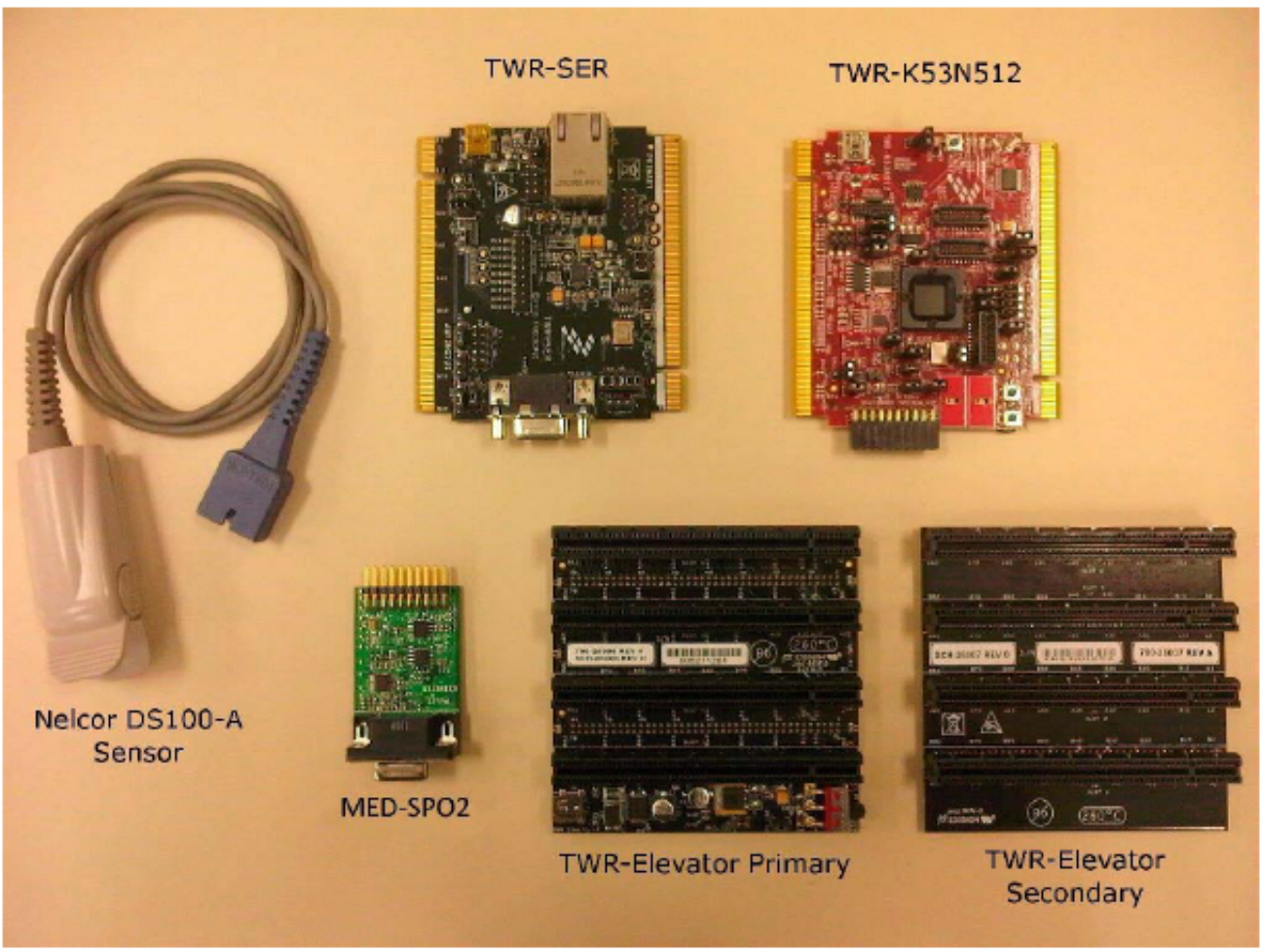

Figure 20. Required components

The TWR-SER and TWR-K53N512 require configuring some jumpers to work properly. The jumper configuration list is the following:

TWR-SER

- Jumper J10 between the pins 1 and 2
- Jumper J16 between the pins 3 and 4

TWR-K53N512

- Jumper J1 Disconnected
- Jumper J3 Disconnected

- Jumper J4 between the pins 2 and 3
- Jumper J11 between the pins 1 and 2

After the jumpers have been configured, the Tower System must be assembled by placing the connector marked as PRIMARY together with the primary elevator, and the connector marked as SECONDARY with the secondary elevator. MED-SPO2 Analog Front End must be connected on the TWR-K53N512 medical connector and the Pulse Oximetry sensor must be connected to the DB9 connector on the MEDSPO2 AFE. Figure 21 shows the Tower System assembled.

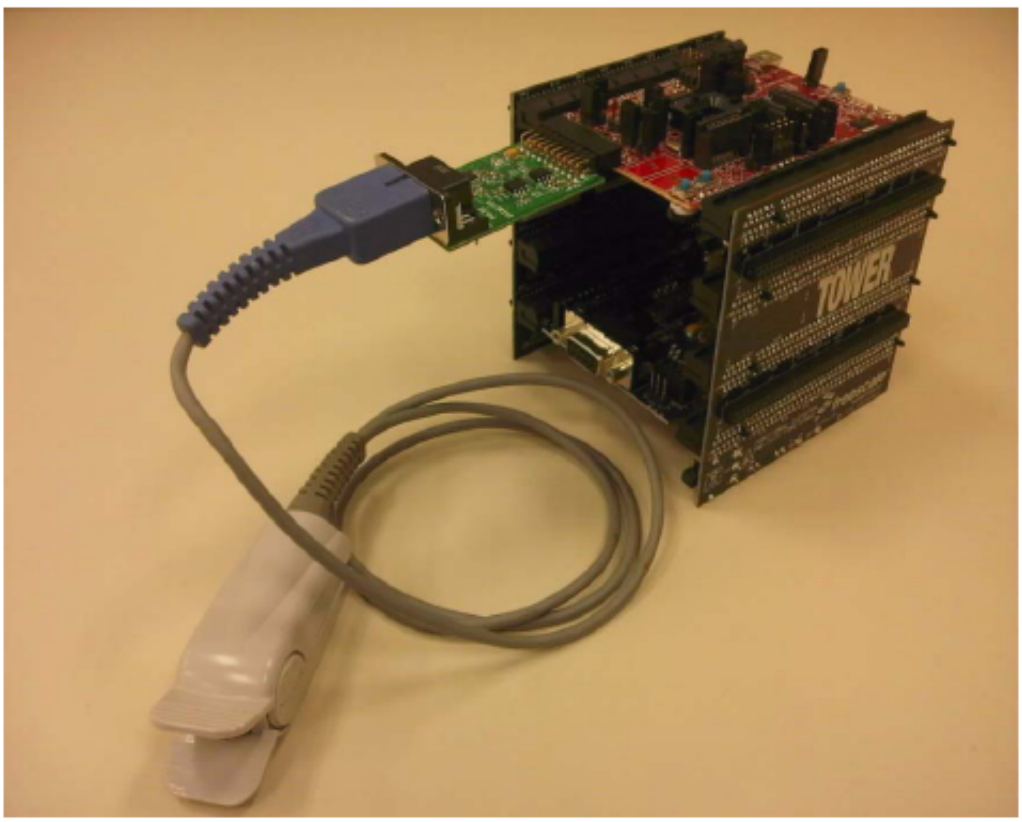

Figure 21. Tower system

5.2 Loading application

The MED-SPO2 program was developed using the IAR Embedded Workbench for ARM v6.1 and can be downloaded from IAR webpage. The name of the project is USB_CDC.eww and can be found on the route \app\cdc\iar_ew\kinetis in the program folder.

1. Connect the USB cable to the TWR-K53N512 board, the device must be recognized as an Open Source BDM—Debug Port.
2. Install the Open Source BDM drivers in the IAR folder.
3. Open the program and verify the debugger in the project options panel by clicking the Menu Project/Options.
4. Select the category Debugger and check that PE micro is selected (Figure 22).

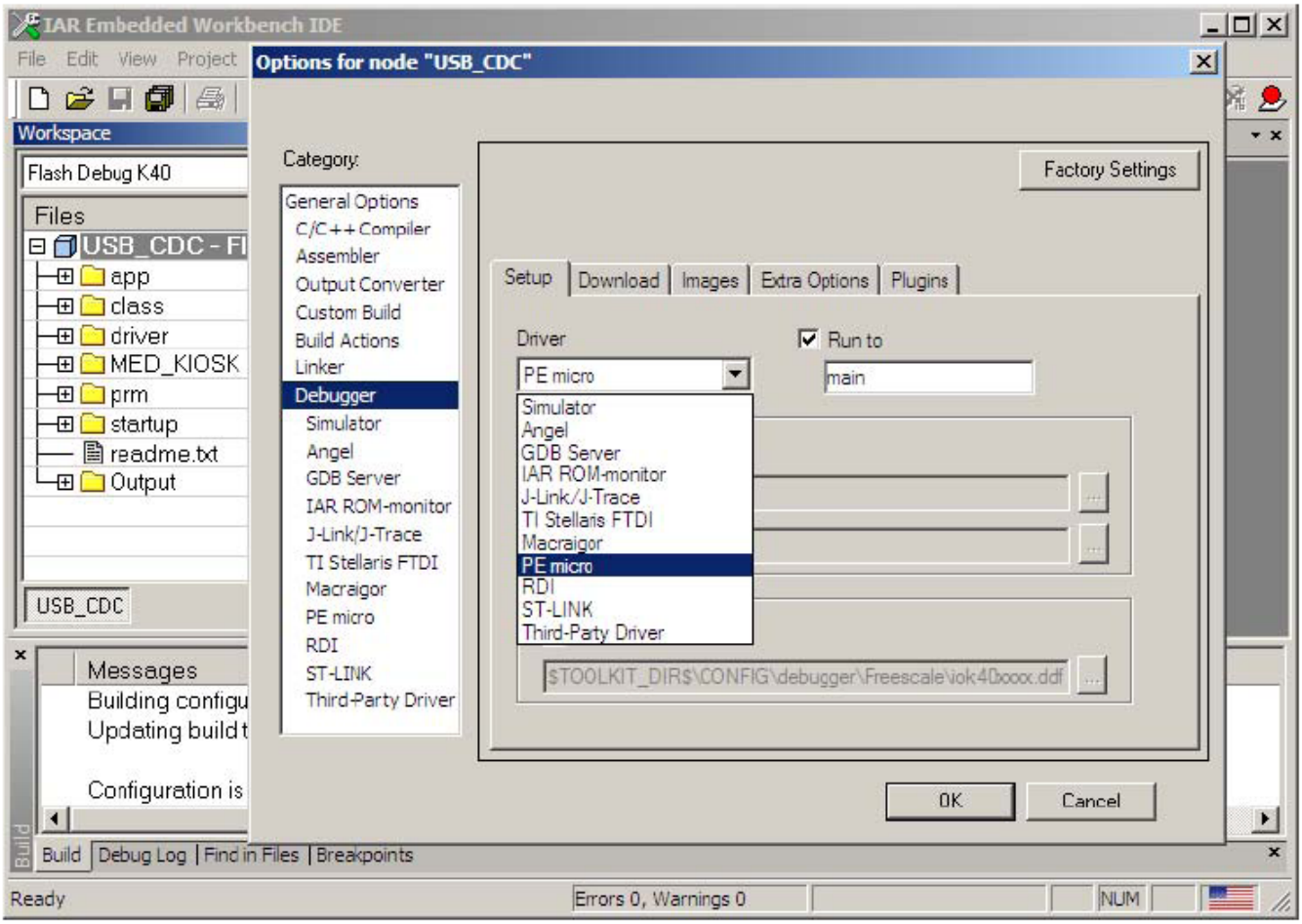

Figure 22. Debugger selection

5. Load the project into the MCU by clicking the DEBUG button (Figure 23). The project will compile automatically and load into the MCU.

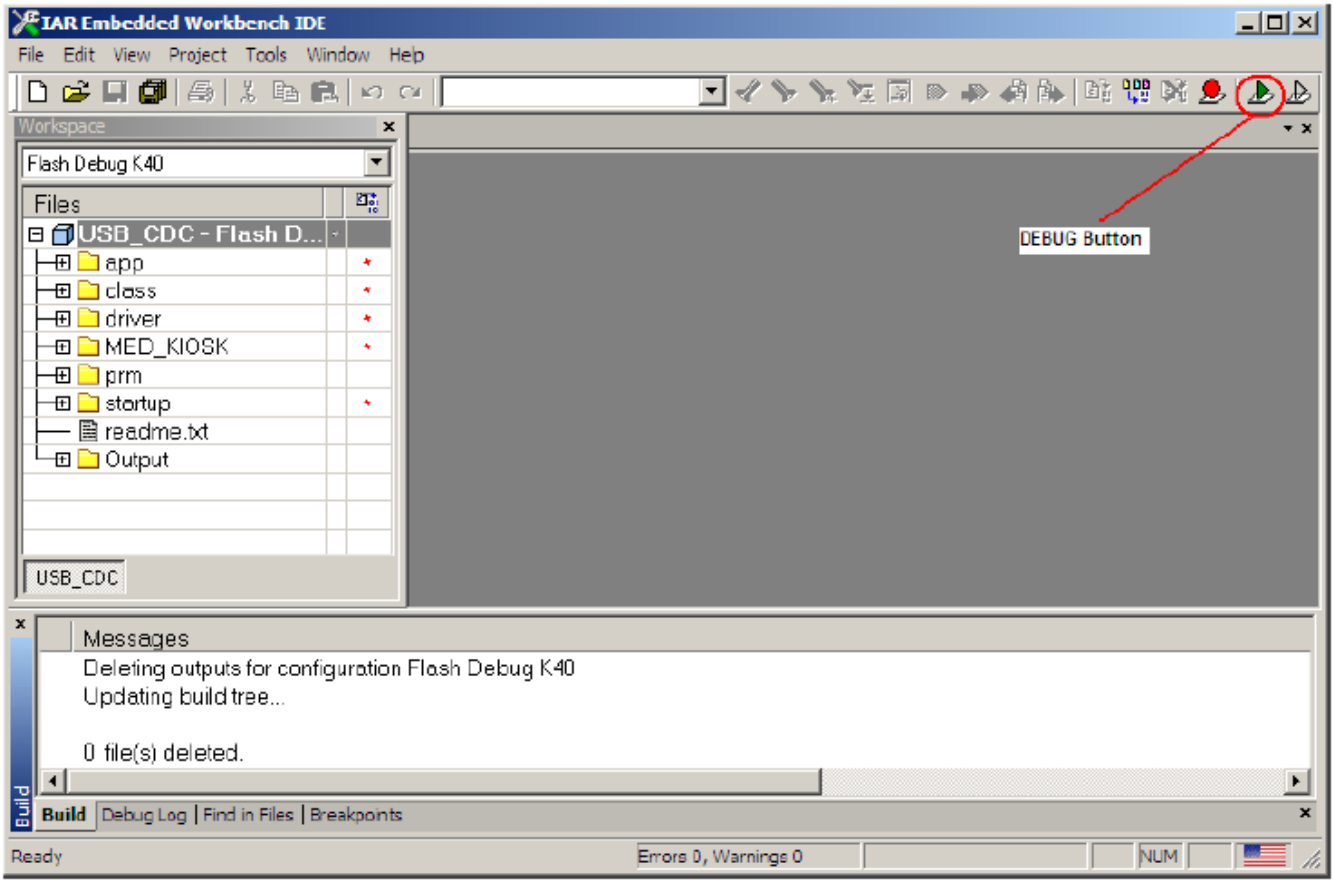

Figure 23. Debug button

5.3 Running demo

After the Tower System has been configured and the project loaded into the MCU, demo can be used. The following steps must be completed to run the demo successfully.

1. Connect the USB cable to the TWR-SER board as shown in Figure 24. The device must be recognized as a Virtual Com Port.

Figure 24. Connecting TWR-SER USB cable

2. Install the driver for the Virtual Com Port included in the project folder after the driver has been installed and device recognized.
3. Open the GUI (Graphic User Interface). A window will appear asking for the COM PORT number assigned to the device.
4. Select the appropriate COM PORT number and click OK (Figure 25), the GUI interface will appear on screen (Figure 26).

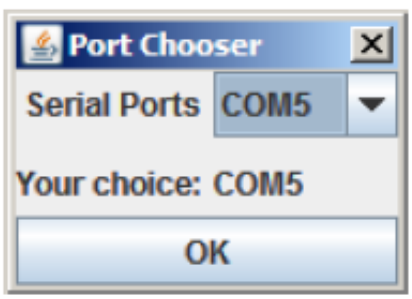

Figure 25. Port chooser

Figure 26. GUI main screen

5. Verify that the Caps Lock is not activated on the keyboard and then press Shift + D to start GUI in doctor mode (Figure 27).

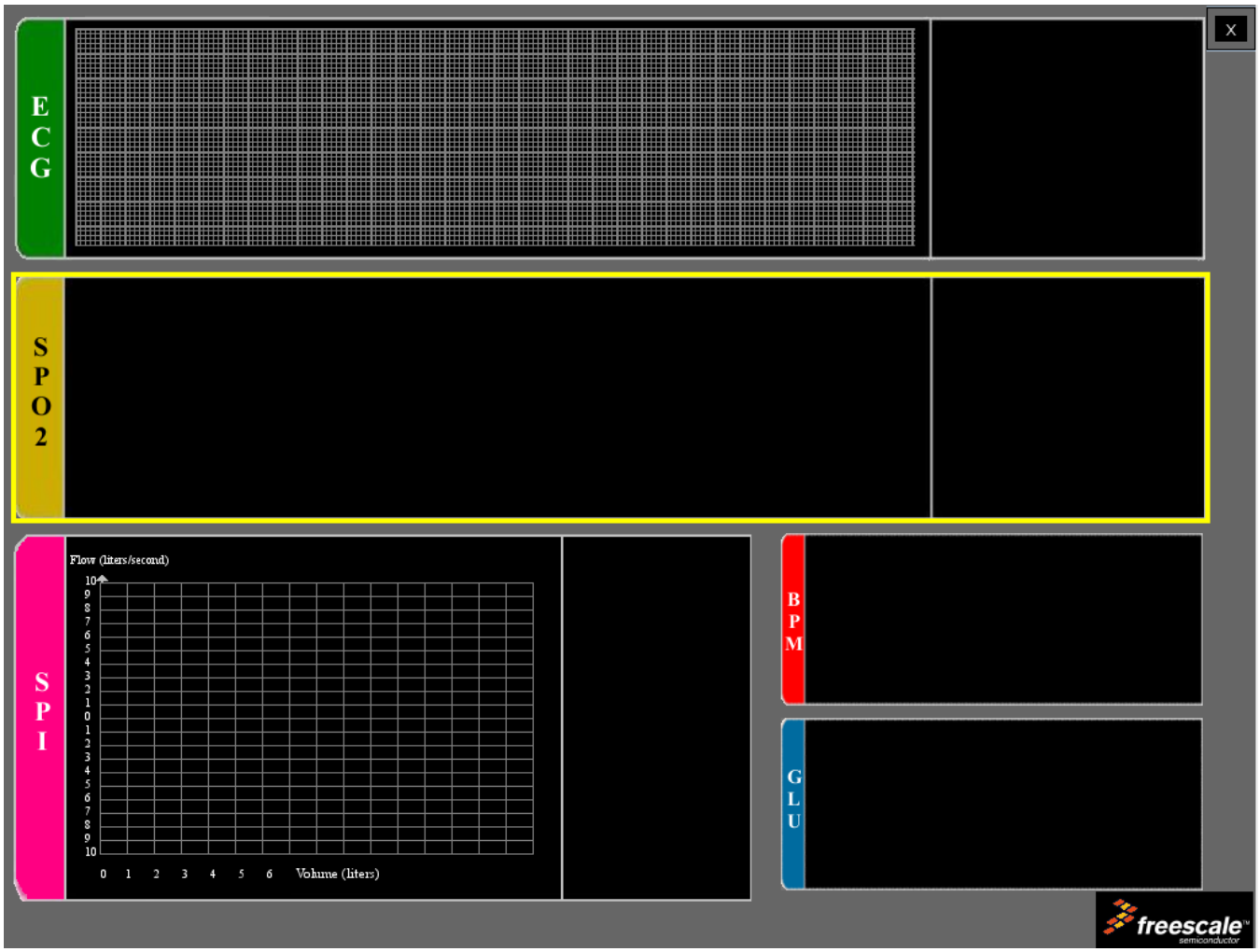

Figure 27. Doctor mode

6. Place the sensor on the finger as shown in Figure 28 and click the SPO2 area on the GUI. The measurements and pulse oximetry graph must appear on the screen (Figure 29).

NOTE

Stay still during this test.

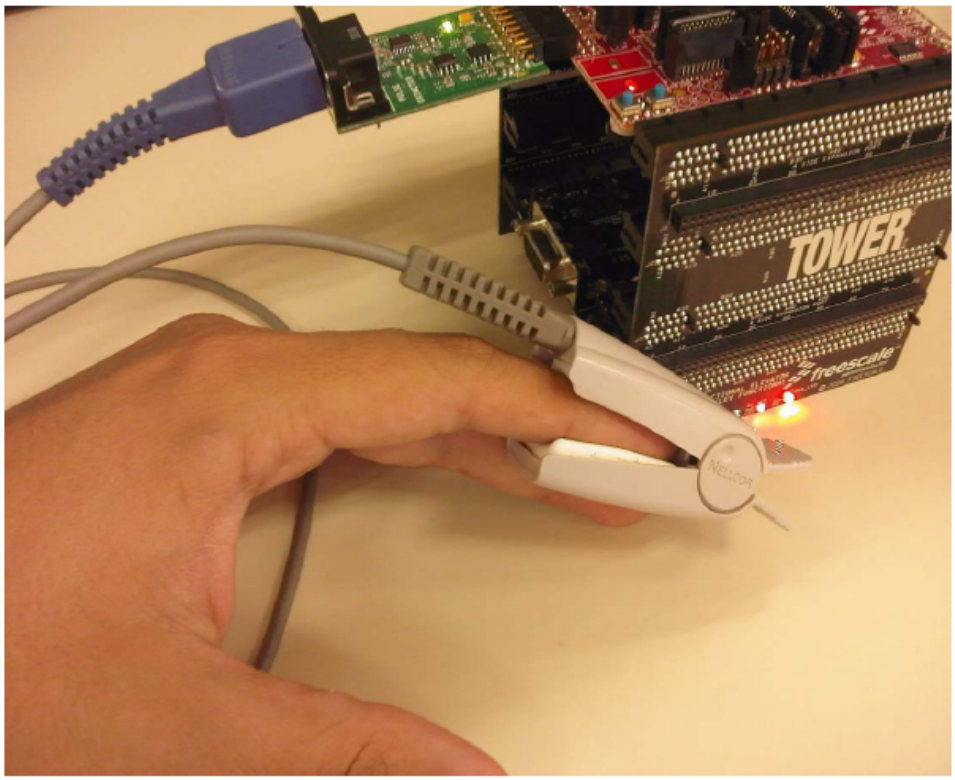

Figure 28. Sensor placement

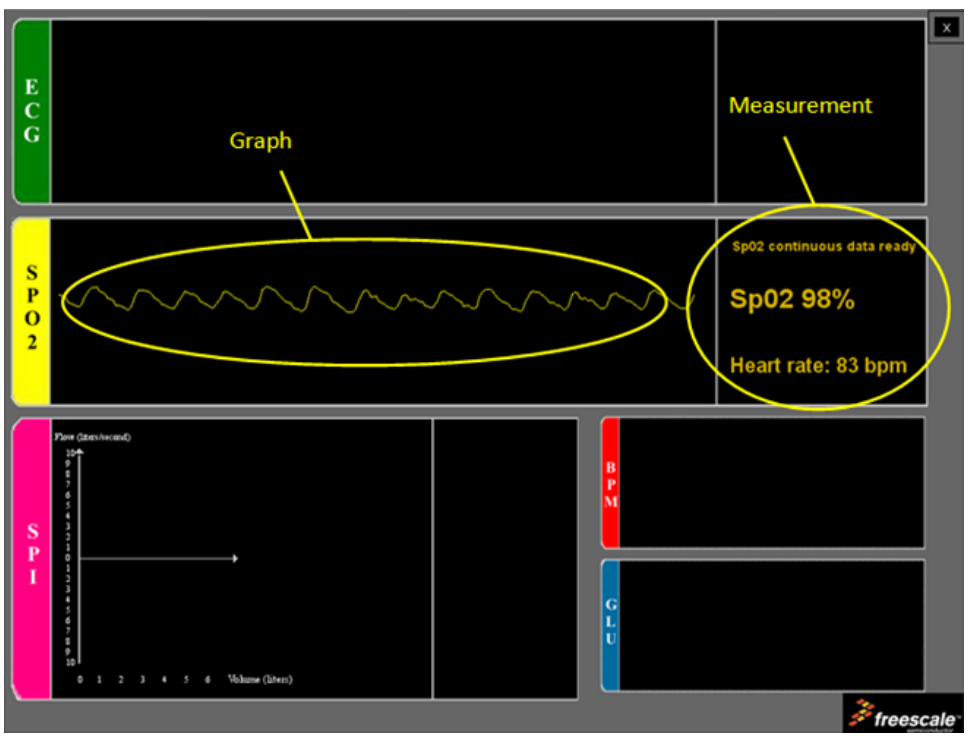

Figure 29. GUI running

6 References

This section contains a list of online resources available on the Freescale website.

- More information about the Kinetis K50 Family can be found on Freescale K50 Family Webpage, **freescale.com/K50_Measurement_MCU**.
- USBAPIRM: Freescale USB Stack with PHDC Device API Reference Manual and useful information can be downloaded from **freescale.com**
- Software was developed and tested on IAR v6.1 for ARM 30 days. Evaluation software can be downloaded from **iar.com/Downloads** (the IAR Webpage in the section Downloads)
- Follow **facebook.com/freescale**, **twitter.com/freescale**, and **youtube.com/freescale**.

7 Conclusions

Freescale Medical portfolio offers a wide range of solutions for medical developments. The Kinetis K50 Family is an example of High Performance, Ultra-Low Power MCUs embedding analog peripherals ideal for bio-signal treatments, and great processing capabilities for digital signal treatment.

The pulse oximeter is an example of the capabilities of the Kinetis K50 Family, which allows developing pulse oximetry applications with only the MCU and a few external components.

Appendix A Software timer

Software timer functions handle an array of subroutines with predefined elapse times. The software timer is constantly checking the elapse times, and when one or more of these times are reached, the respective subroutine is called.

The software timer allows better management of time dependent subroutines because only one MCU timer is needed. The timer must be configured to generate 1 ms interruptions and a variable must be increased by one on every interrupt execution indicating the quantity of milliseconds elapsed.

A.1 Initializing software timer

An MCU timer must be initialized to generate an interrupt every 1 ms. On the interrupt routine a global variable must be increased by one on every interrupt execution. This variable must be specified on the file SwTimer.h and the initialization function must be called on the function SwTimer_Init in the file SwTimer.c.

At the beginning of the application, function SwTimer_Init must be called. This function cleans the objects array, releasing all the software timers and leaving them available for application. The MCU timer initialization must be called in this function.

A.2 Creating a software timer

After the Software Timer has been initialized, a Timer can be created by using the function SwTimer_CreateTimer(pFunc_t callBackFunc). The input parameter is the name of the function to be called. When this function is executed, it returns a timerId value that is necessary to start or stop the created timer. If a 0xFF is returned as timerId, this means that the maximum number of timers has been reached and the timer could not be initialized. The maximum number of timers is defined as MAX_TIMER_OBJECTS and can be found on file SwTimer.h.

Example: My_Timer_Id = SwTimer_CreateTimer(Function_To_Be_Called);

Function SwTimer_StartTimer(UINT8 timerId, UINT16 tickPeriod_ms) starts the SwTimer, the input parameter timerId is the timerId number returned by the Create Timer function when the timer was created. tickPeriod_ms is the period of time in milliseconds that has to elapse to execute the function. The following example starts the previous created timer to execute Function_To_Be_Called every 10 ms.

Example: SwTimer_StartTimer(My_Timer_Id, 10);

When the time defined has elapsed, the function Function_To_Be_Called is executed and the timer is deactivated. A new Start Timer statement must be written to activate the timer again, when the programmed time has elapsed. Function SwTimer_StopTimer(UINT8 timerId) stops the selected timer.

A.3 Functional description

When the function SwTimer_CreateTimer is called, a new object in the Timer Object Array is created. Function SwTimer_PeriodicTask is constantly called on the Main Application Routine and checks all the timer objects on the Timer Object Array. When the MCU timer, configured to interrupt every 1 ms, generates an interrupt, a variable increases its count by one indicating that 1 ms has elapsed. The SwTimer_PeriodicTask function checks this variable. If it is different than zero, the value of this variable is taken away from the Timer Object programmed Time. If the programmed time of a Timer Object reaches zero, the subroutine declared for that timer on the function SwTimer_CreateTimer is called and the created times is set to INACTIVE until a new SwTimer_StartTimer is called.

The following block diagram shows the SwTimer_PeriodicTask subroutine.

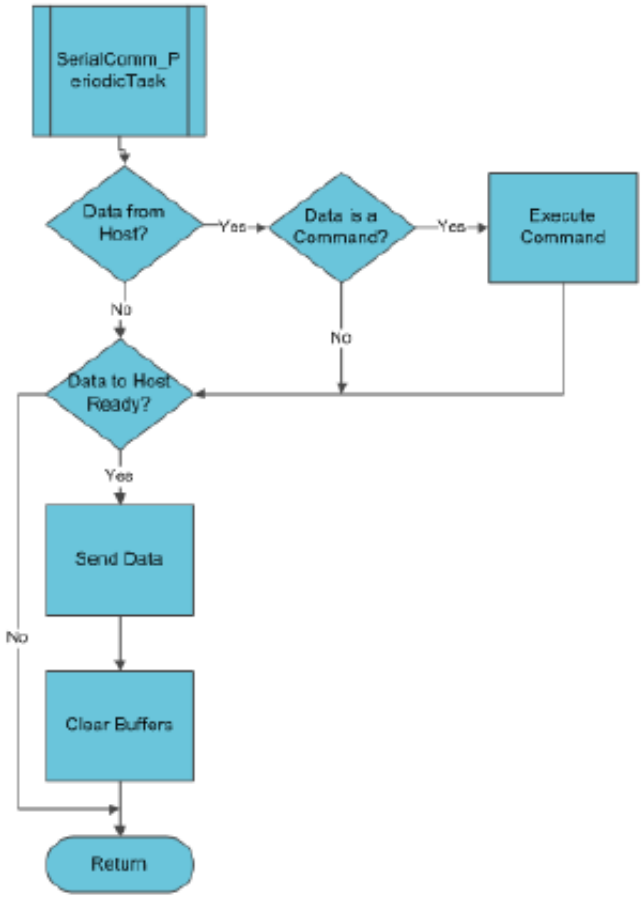

Figure A-1. Block diagram

Appendix B Communication protocol

The application communicates with the GUI (Graphic User Interface) in the PC using the Freescale USB Stack with PHDC with the device acting as a CDC (Communication Device Class). The device communicates via a serial interface similar to the RS232 communications standard, but emulating a virtual COM port.

After the device has been connected and a proper driver has been installed, the PC recognizes it as a Virtual COM Port and it is accessible, for example using HyperTerminal. Communication is established using the following parameters.

- Bits per second—115,200
- Data bits—8
- Parity—None
- Stop Bits—1
- Flow Control — None

Communication starts when the host (PC) sends a request packet indicating to the device the action to perform. The device then responds with a confirmation packet indicating to host that the command has been received. At this point, the host must be prepared to receive data packets from the device and show the data received on the GUI. Communication finishes when the host sends a request packet indicating the device to stop. The following block diagram describes the data flow.

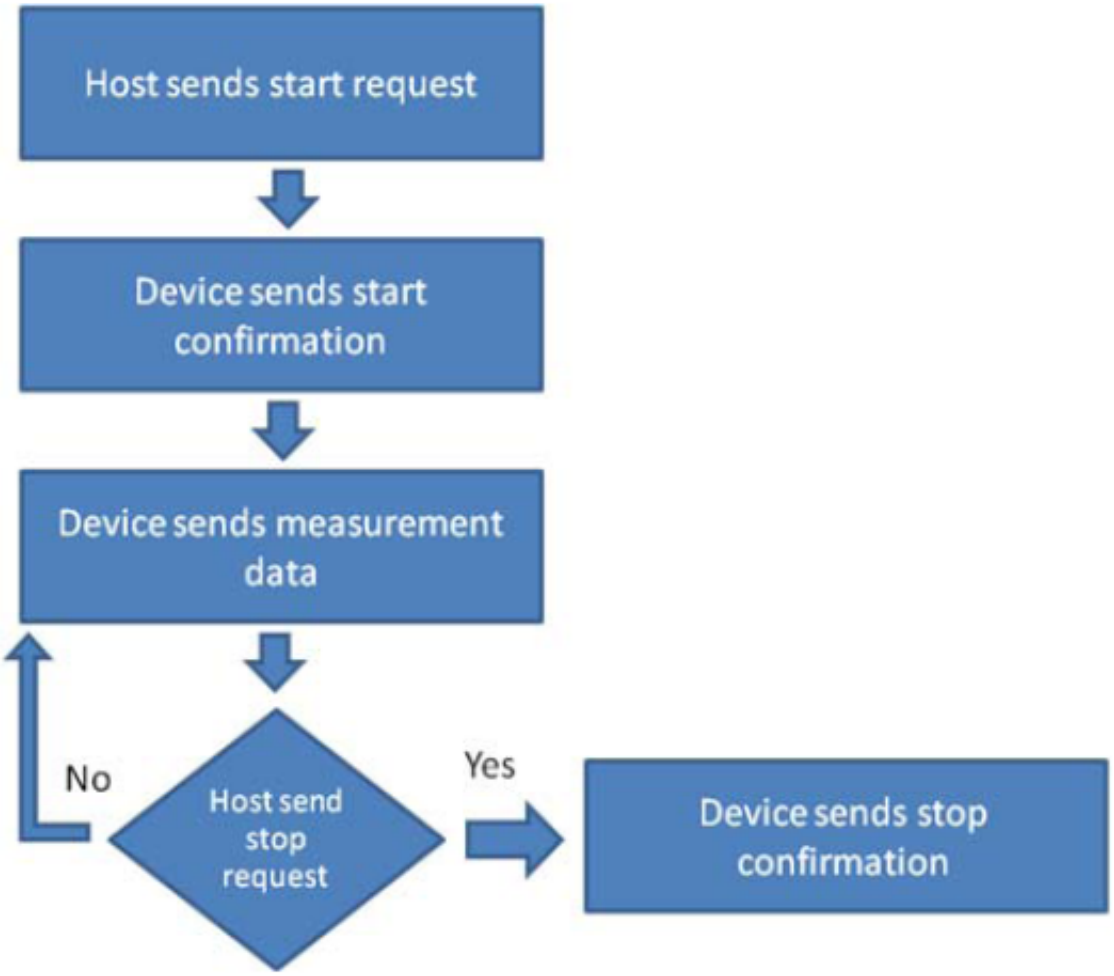

Figure B-1. Communication protocol data flow

Packets sent between host and device have a specific structure. The Packet is divided in four main parts:

- Packet Type
- Command Opcode
- Data length
- Data

The image below shows the packet structure.

Figure B-2. Packet structure

B.1 Packet type

The Packet Type byte defines the kind of packet sent. There are three kinds of packets that can be sent between host and device.

B.1.1 REQ packet

This is a request packet, this kind of packet is used by the host to request to the device to perform some action like a start or stop measurement. A REQ packet is usually composed of 2 bytes, Packet Type and Command Opcode. Data Length and Data Packet bytes are not required.

B.1.2 CFM packet

This is a confirmation packet. This kind of packet is used by the device to confirm to the host that a command has been received, and sends a response indicating if the command is accepted, or if the device is busy.

B.1.3 IND packet

This is an indication packet. This kind of packet is used to indicate to the host that an event has occurred in the device and data needs to be sent. For example, this is used when the device has a new data to be sent to the GUI.

The following table shows the HEX codes for every Packet Type.

Table B-1. HEX codes

Packet type	Hex codes
REQ	0x52
CFM	0x43
IND	0x69

B.2 Command opcode

The Command Opcode byte indicates the action performed for a REQ packet, and the kind of confirmation or indication, in case of CFM and IND packet types. There are different Opcodes for every Packet Type. The following table shows the different Opcodes (See Note below).

Table B-2. Opcodes

Opcode	REQ	CFM	IND	Opcodes (Hex)
Glucose meter				
GLU_START_MEASUREMENT	X	X		0x00
GLU_ABORT_MEASUREMENT	X	X		0x01
GLU_START_CALIBRATION	X	X		0x02

Table continues on the next page...

Table B-2. Opcodes (continued)

Opcode	REQ	CFM	IND	Opcodes (Hex)
GLU_BLOOD_DETECTED			X	0x03
GLU_MEASUREMENT_COMPLETE_OK			X	0x04
GLU_CALIBRATION_COMPLETE_OK			X	0x05
Blood Pressure Meter				
BPM_START_MEASUREMENT	X	X		0x06
BPM_ABORT_MEASUREMENT	X	X		0x07
BPM_MEASUREMENT_COMPLETE_OK			X	0x08
BPM_MEASUREMENT_ERROR			X	0x09
BPM_START_LEAK_TEST	X	X		0x0A
BPM_ABORT_LEAK_TEST	X	X		0x0B
BPM_LEAK_TEST_COMPLETE			X	0x0C
BPM_SEND_PRESSURE_VALUE_TO_PC			X	0x28
Electro Cardiograph Opcode				
ECG_HEART_RATE_START_MEASUREMENT	X	X		0x0D
ECG_HEART_RATE_ABORT_MEASUREMENT	X	X		0x0E
ECG_HEART_RATE_MEASUREMENT_COMPLETE_OK			X	0x0F
ECG_HEART_RATE_MEASUREMENT_ERROR			X	0x10
ECG_DIAGNOSTIC_MODE_START_MEASUREMENT	X	X		0x12
ECG_DIAGNOSTIC_MODE_STOP_MEASUREMENT	X	X		0x13
ECG_DIAGNOSTIC_MODE_NEW_DATA_READY			X	0x14
Thermometer				
TMP_READ_TEMEPRATURE	X	X		0x15
Height scale				
HGT_READ_HEIGHT	X	X		0x16
Weight scale				
WGT_READ_WEIGHT	X	X		0x17
Spirometer				
SPR_DIAGNOSTIC_MODE_START_MEASURMENT	X	X		0x0C
SPR_DIAGNOSTIC_MODE_STOP_MEASURMENT	X	X		0x0D
SPR_DIAGNOSTIC_MODE_NEW_DATA_READY			X	0x0E
Pulse oximetry				
SPO2_START_MEASURMENT	X	X		0x21
SPO2_ABORT_MEASURMENT	X	X		0x22
SPO2_MEASURMENT_COMPLETE_OK			X	0x23
SPO2_MEASURMENT_ERROR			X	0x24
SPO2_DIAGNOSTIC_MODE_START_MEASURMENT	X	X		0x25
SPO2_DIAGNOSTIC_MODE _STOP_MEASURMENT	X	X		0x26

Table continues on the next page...

Table B-2. Opcodes (continued)

Opcode	REQ	CFM	IND	Opcodes (Hex)
SPO2_DIAGNOSTIC_MODE_NEW_DATA_READY			X	0x27
				0x21
System commands				
SYS_CHECK_DEVICE_CONNECTION	X	X		0x29
SYS_RESTART_SYSTEM	X			0x2A

NOTE

Software related with this application note does not respond to all of these commands.

B.3 Data length and data string

The data length and data string bytes are the data quantity count and the data itself. The data length byte represents the number of bytes contained into the data string. The data string is the information sent, just the data, therefore the Data Length byte must not count the Packet Type byte, the Command Opcode byte or itself.

B.4 Functional description

Communication starts when the host sends a REQ packet indicating to the device to start a new measurement. The host must send a REQ Packet Type to start transactions (Figure B-3).

Figure B-3. Start packet sent by host

The Start Opcode can be any Opcode related with start a measurement, for example, if we wanted to start the ECG in diagnostic mode, the Data Packet will look like Figure B-4.

Figure B-4. Starting ECG in diagnostic mode

0x52 is the HEX code for a request (REQ) Packet Type, 0x12 corresponds to ECG_DIAGNOSTIC_MODE_START_MEASUREMENT. After sending the REQ packet, a CFM packet must be received indicating the status of the device. The received packet must look like Figure B-5.

Figure B-5. Confirmation packet structure

The error byte indicates the device status. The table below shows possible error codes.

Table B-3. Error codes

Error	HEX code
OK	0x00
BUSY	0x01
INVALID OPCODE	0x02

If the error byte received corresponds to OK, the device starts sending data as soon as a new data packet is ready. If BUSY error is received, the host must try to communicate later. If the error received is INVALID OPCODE, data sent and transmission lines must be checked.

If a CFM packet with an OK error has been received, the device starts sending Information related with the measurement requested. This is performed using indication packets (IND). Indication packet structure is shown in Figure B-6.

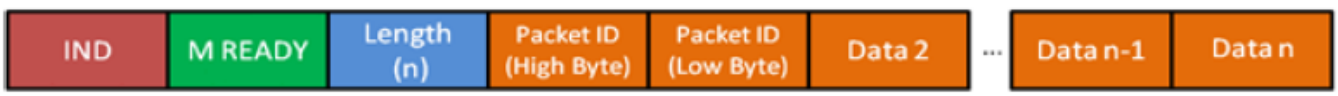

Figure B-6. Indication packet structure

The first byte contains the HEX code for an Indication Packet type. The second byte contains the Opcode for the kind of indication, for example if the device is sending an Indication Packet for ECG_DIAGNOSTIC_MODE_NEW_DATA_READY, the HEX code read in this position is 0x14 because this is the Indication Opcode for a new set of data from the ECG diagnostic mode. The next byte is the Length which indicates the quantity of data sent.

The first couple of bytes after the Length byte are the Packet ID bytes. The Packet ID is a 16-bit data divided in 2 bytes to be sent and contains the number of packets sent. The Packet ID number of a data packet is the Packet ID of the previous packet + 1. For example, if the Packet ID of the previous packet sent was 0x0009, the Packet ID of the next packet must be 0x000A. This allows the GUI to determine if a packet is missing.

The following data bytes are the Data String and contain the information of the measurement requested. The Data quantity is determined by the Data Length byte and data is interpreted depending on the kind of measurement. For example, for the MED-EKG Demo from Data 2 to Data n-1 contains the data graphed. Every point in the graph is represented by a 16-bit signed number, this means that every 2 data bytes in the packet, means it is one point in the graph. The first byte is the most significant part of the long (16-bits) and the second byte is the less significant part. The long is signed using 16-bit complement. The last byte contains the Heart Rate measurement. This byte must be taken as it is, an unsigned char data that contains the number of beats per minute. Figure B-7 shows a typical MED-EKG demo indication packet.

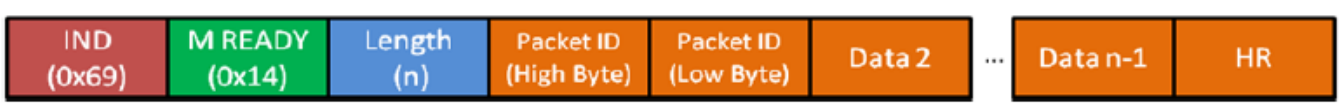

Figure B-7. MED-EKG IND packet

When a Stop request is sent by the host, the device stops sending data and waits for a new Start request command. Figure B-8 shows the Stop Command structure.

Figure B-8. Stop command structure

Immediately after this, the device must acknowledge with a CFM packet shown in Figure B-9.

Figure B-9. Stop CFM packet

The CFM packet for stop does not require an error code, it just must be received. If this packet has not been received, the request has been rejected or not taken and must be sent again to stop the measurements.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: AN4327
Rev. 2, 11/2012