

# Blood Pressure Monitor Fundamentals and Design

by: **Santiago Lopez**

## Contents

## 1 Introduction

This application note demonstrates the implementation of a basic blood pressure monitor using Freescale products. The blood pressure monitor can be implemented using any of the Freescale medical oriented MCUs: Kinetis MK53N512 and Flexis MM members MC9S08MM128 and MCF51MM256 embedding a 16-bit ADC, 12-bit DAC, 2 Programmable-Gain Op-Amps, 2 TRIAMPS, Analog Comparators, and Vref generator. The K50 family can also perform DSP instructions for signal treatment and MCF51MM can perform multiply and accumulate (MAC) instructions.

This document is intended to be used by biomedical engineers, medical equipment developers, or any person related with the practice of medicine and interested in understanding the operation of blood pressure monitors. Nevertheless, it is necessary to know fundamentals of electronic, analog, and digital circuits.

## 2 Blood pressure fundamentals

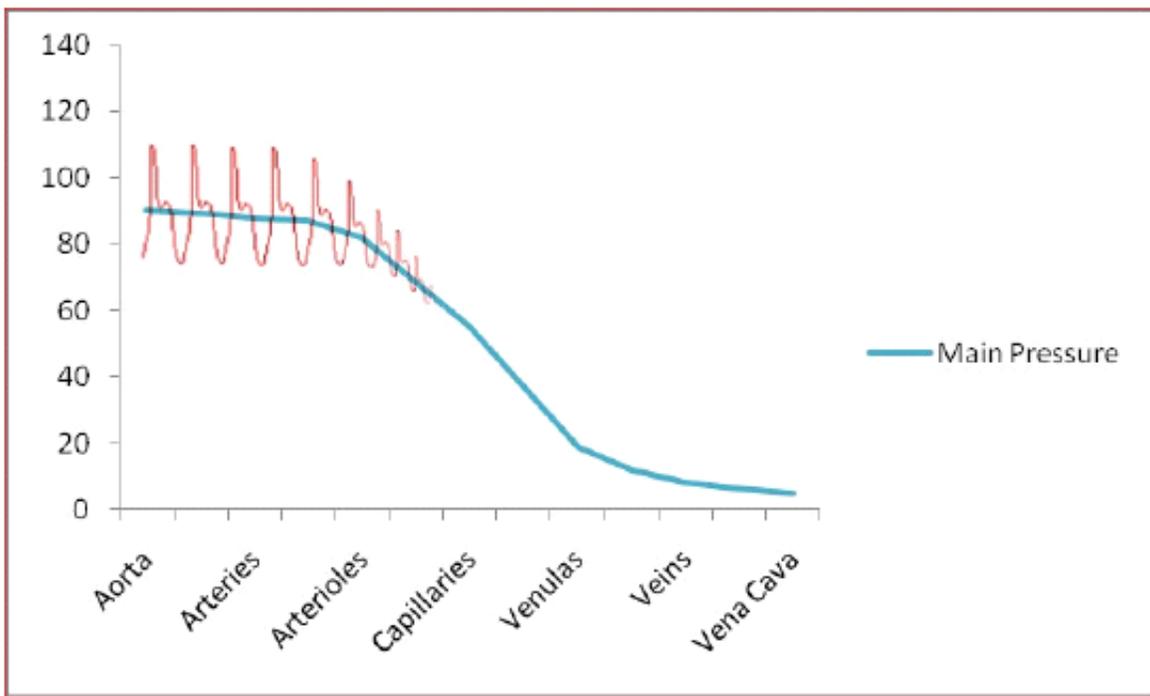
This section contains information about physiological concepts of arterial pressure and blood pressure monitor operating principle.

1	Introduction.....	1
2	Blood pressure fundamentals.....	1
3	Blood pressure monitor implementation.....	3
4	Software model.....	8
5	Running blood pressure monitor demo .....	16
6	References.....	29
7	Conclusions.....	29
A	Software timer.....	30
B	Communication protocol.....	32

## 2.1 Arterial pressure

Arterial pressure is defined as the hydrostatic pressure exerted by the blood over the arteries as a result of the heart left ventricle contraction. Systolic arterial pressure is the higher blood pressure reached by the arteries during systole (ventricular contraction), and diastolic arterial pressure is the lowest blood pressure reached during diastole (ventricular relaxation). In a healthy young adult at rest, systolic arterial pressure is around 110 mmHg and diastolic arterial pressure is around 70 mmHg.

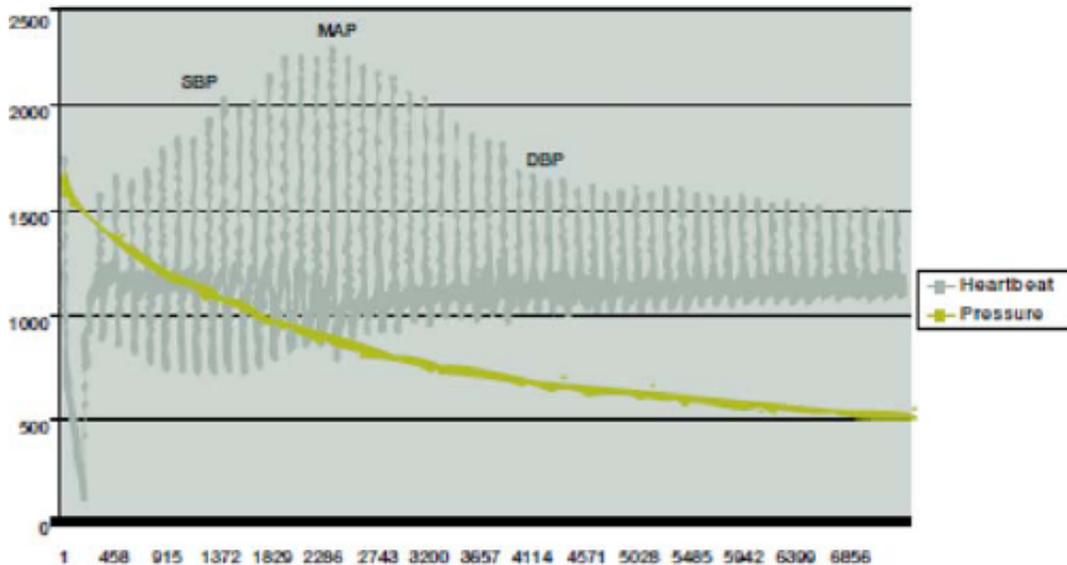
Blood flow is the blood volume that flows through any tissue in a determined period of time (typically represented as ml/min) in order to bring tissue oxygen and nutrients transported in blood. Blood flow is directly affected by the blood pressure as blood flows from the area with more pressure to the area with less pressure. Greater the pressure difference, higher is the blood flow. Blood is pumped from the left ventricle of the heart out to the aorta where it reaches its higher pressure levels. Blood pressure falls as blood moves away from the left ventricle until it reaches 0 mm Hg, when it returns to the heart's right atrium. [Figure 1](#) represents pressure changes.



**Figure 1. Pressure changes on blood vessels**

## 2.2 Blood pressure monitor operating principle

Blood pressure monitor operation is based on the oscillometric method. This method takes advantage of the pressure pulsations taken during measurements. An occluding cuff is placed on the left arm and is connected to an air pump and a pressure sensor. Cuff is inflated until a pressure greater than the typical systolic value is reached, then the cuff is slowly deflated. As the cuff deflates, when systolic pressure value approaches, pulsations start to appear. These pulsations represent the pressure changes due to heart ventricle contraction and can be used to calculate the heartbeat rate. Pulsations grow in amplitude until mean arterial pressure (MAP) is reached, then decrease until they disappear. [Figure 2](#) shows the cuff pressure vs. pulsations.



**Figure 2. Cuff pressure vs. heartbeat signal**

Oscillometric method determines the MAP by taking the cuff pressure when the pulse with the largest amplitude appears. Systolic and diastolic values are calculated using algorithms that vary among different medical equipment developers. Freescale Blood Pressure Monitor calculates the systolic and diastolic pressure by considering that systolic pressure is approximately equal to the pressure measurement taken in the cuff when a pulse with 70% of the amplitude of the MAP pulse appears while the cuff pressure is above the MAP value. Diastolic pressure is approximately equal to the cuff pressure value registered when a pulse with 50% of the MAP pulse amplitude appears while the cuff pressure is under the MAP value.

### 3 Blood pressure monitor implementation

Blood pressure monitor is implemented using Freescale medical-oriented Kinetis K53 MCUs and Flexis MM devices, which feature the following characteristics:

- 16-bit ADC
- 12-bit DAC
- 2x programmable gain operational amplifiers (OpAmps)
- 2x transimpedance amplifiers (TRIAMPS)
- $V_{ref}$  generator
- Set of DSP instructions including MAC (Only K5X Family)
- Multiply and Accumulate (MAC) instruction on MCF51MM

Freescale medical-oriented MCUs reduce the Bill Of Materials (BOM) required for medical applications and provide great processing capabilities ideal for medical equipment. Nevertheless, some external circuitry is needed for pressure sensing and cuff control.

#### 3.1 MED-BPM analog front end

MED-BPM Analog Front End (AFE) is a demo board designed for work as a blood pressure monitor in conjunction with a Freescale medical-oriented MCU. MED-BPM communicates with the MCU using the medical connector, and allows for easy prototyping and reduced time to market by using the Freescale Tower System. MED-BPM block diagram is shown below ([MED-BPM analog front end](#)).

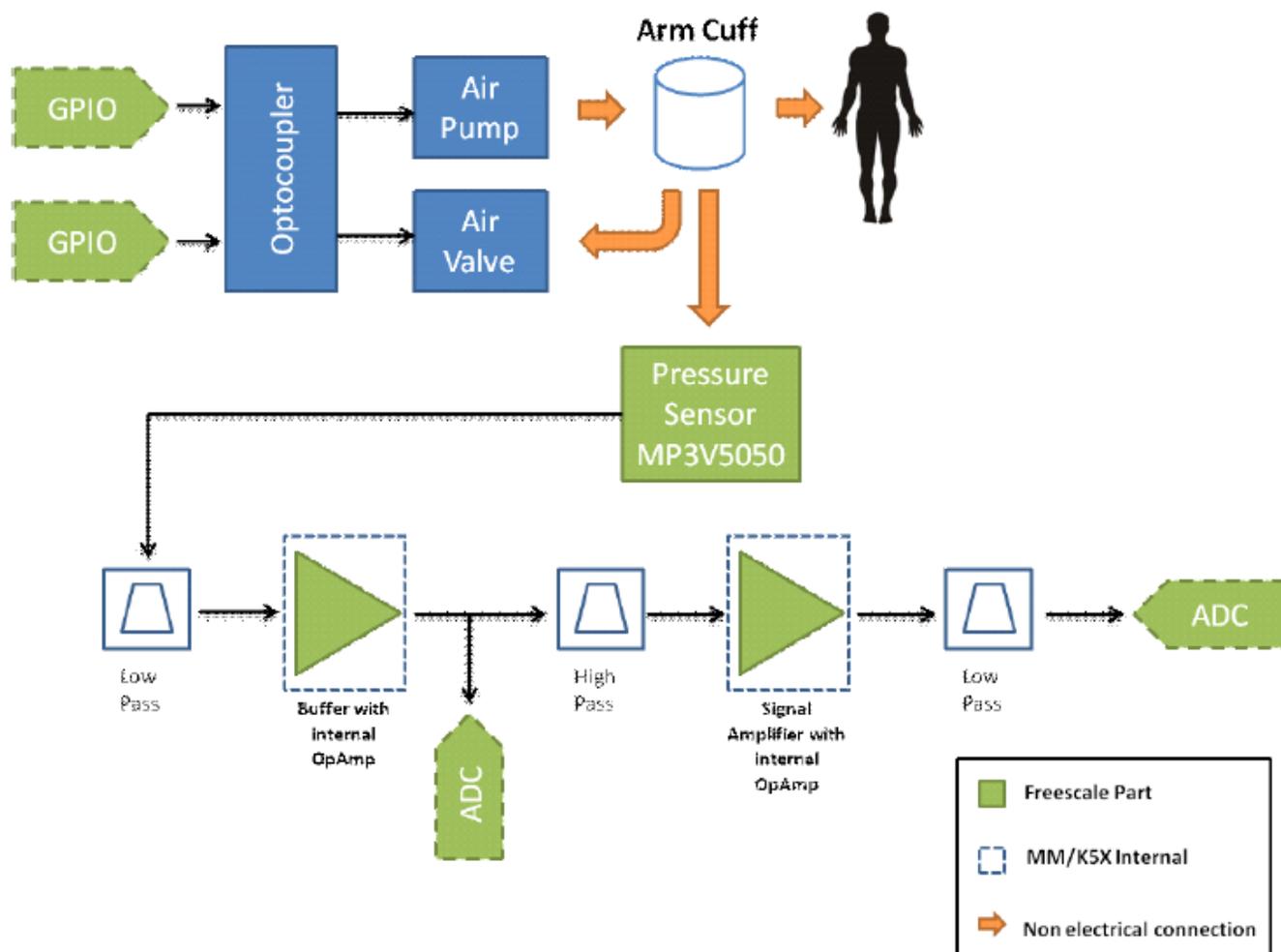


Figure 3. MED-BPM block diagram

### 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<sup>2</sup>C 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
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

Table continues on the next page...

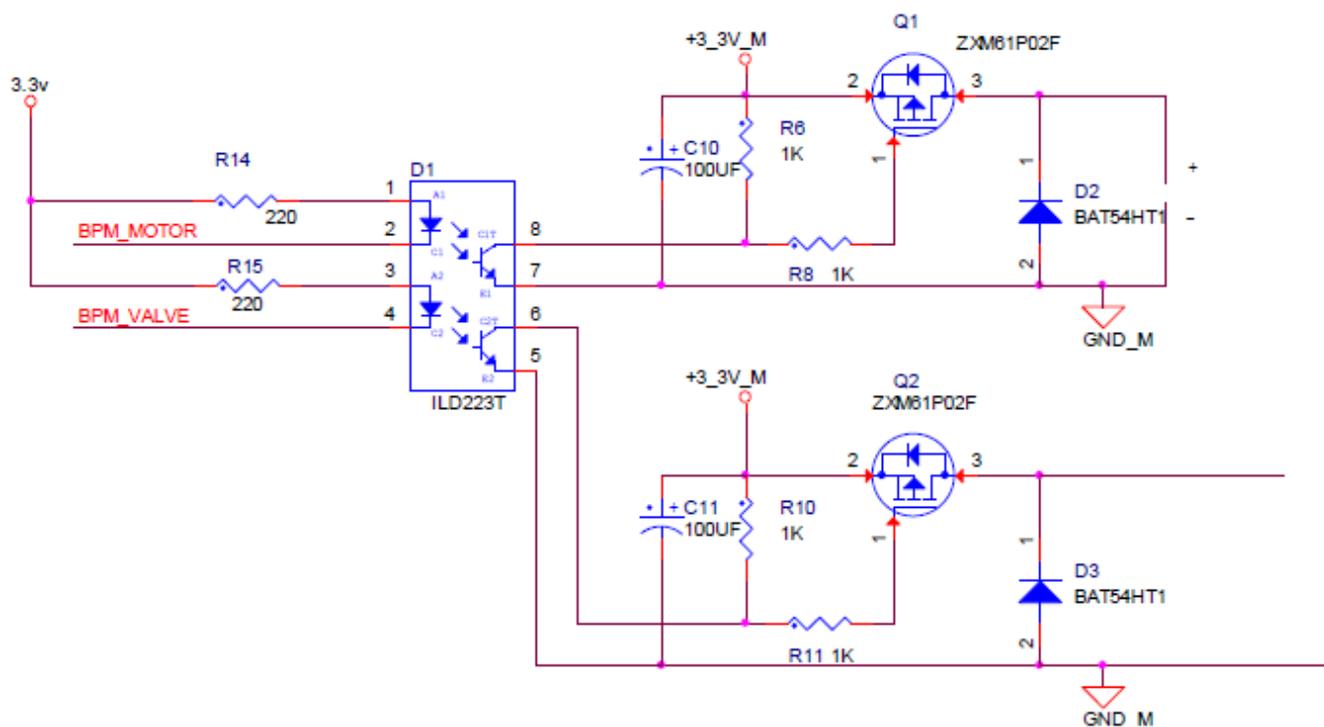
**Table 1. Medical connector signals (continued)**

19	TRIAMP 1 Out	TRIAMP 2 Out	20
----	--------------	--------------	----

### 3.1.2 Arm cuff pressure control

MED-BPM works using an oscillometric method for blood pressure measurements. This is a noninvasive method which requires an external arm cuff in order to occlude the patient’s arm and detect the systolic and diastolic arterial pressure. The arm cuff is inflated using an external air pump controlled with an MCU GPIO pin, and deflated by activating an escape valve with another GPIO pin.

Because the current provided by the USB port (500 mA) is not enough to activate the air pump and the valve (600 mA), those external components are activated by using an external power source which provides sufficient current. An optocoupler is needed for coupling MCU control signals with the components to activate. Figure 4 shows the coupling stage.



**Figure 4. Coupling stage**

Output from the optocoupler is connected to a MOSFET working as a switch, so the air pump and valve mechanisms can be activated successfully.

### 3.1.3 External elements connector

After the optocoupling and switching stage, a connector for an external air pump, escape valve, and batteries is placed on the MED-BPM board. This allows the control of external components using MCU signals. The air pump motor and escape valve are powered using two external AA 1.5 V batteries, because USB output cannot provide enough current to drive those devices. Figure 5 shows the placement of the connector pins.

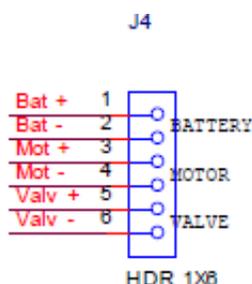


Figure 5. External components connector

### 3.1.4 Pressure sensor

The functionality of the oscillometric method is based on the measurement of the pressure variations in the arm cuff. Pressure in the cuff is measured by using the Freescale Pressure Sensor MP3V5050 which integrates on-chip, bipolar OpAmp circuitry and thin film resistor networks to provide a high output signal and temperature compensation. Main characteristics of the MP3V5050 are featured in the following table

Table 2. MPV3V5050 main characteristics

Characteristic	Value	Unit
Pressure range	1–50	kPa
Supply voltage	2.7–3.3	V <sub>DC</sub>
Accuracy	±2.5	%V <sub>FSS</sub>
Sensitivity	54	mV/kPa

MP3V5050 delivers a voltage proportional to the input pressure. This sensor is directly connected to the amplification stage. More information can be found on [freescale.com](http://freescale.com).

### 3.1.5 Signal filtering and amplification

This stage is composed of three filters, one buffer circuit, and one non-inverting amplifier (Figure 6). Filters are first order RC passive type, and the cut-off frequency is described by the following formula.

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

1. A signal is passed through a 10 Hz RC low-pass filter (LPF) composed of a resistor and a capacitor in order to remove high-frequency noise.
2. Then the signal is passed through a buffer circuit consisting of a single Op-Amp in buffer mode to couple the signal to the sensor. The output from the buffer circuit is where the arterial pressure measurements are taken.
3. The signal is then filtered again with a 2.2 Hz RC high-pass filter which removes high-frequency noise and gets a cleaner signal for amplification.
4. This signal is amplified using a non-inverting amplifier composed by a second Op-Amp and two resistors, (100 kΩ and 1 kΩ) generating a gain of 101 so cuff oscillations can be distinguished better.
5. After this stage, the signal is filtered again with another 10 Hz RC LPF so high-frequency noise can be removed.

Figure 6 shows the filtering and amplification stage.

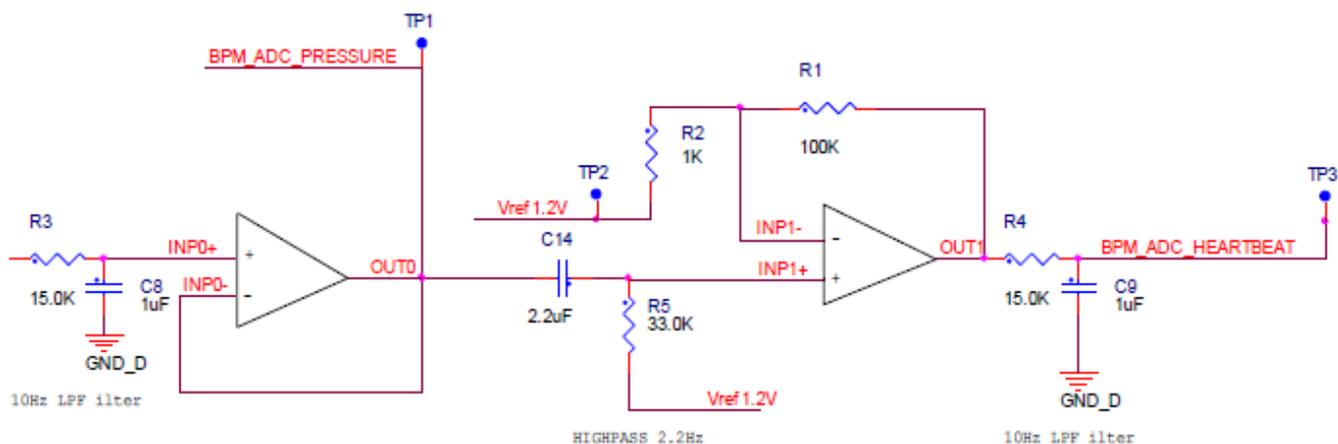


Figure 6. Filtering and amplification circuit

### 3.2 Functional description

MED-BPM demo uses a variation of the oscillometric method called Ramp-Up method that takes measurements while the cuff is inflating.

In the Ramp-Up method, an inflatable air cuff is placed on the patient’s left arm and is adjusted so it is tight around the arm. The escape valve is closed and the air pump starts to inflate the cuff. While inflating, the main pressure in the cuff is monitored and amplified in order to get the cuff pressure oscillations (Figure 7).

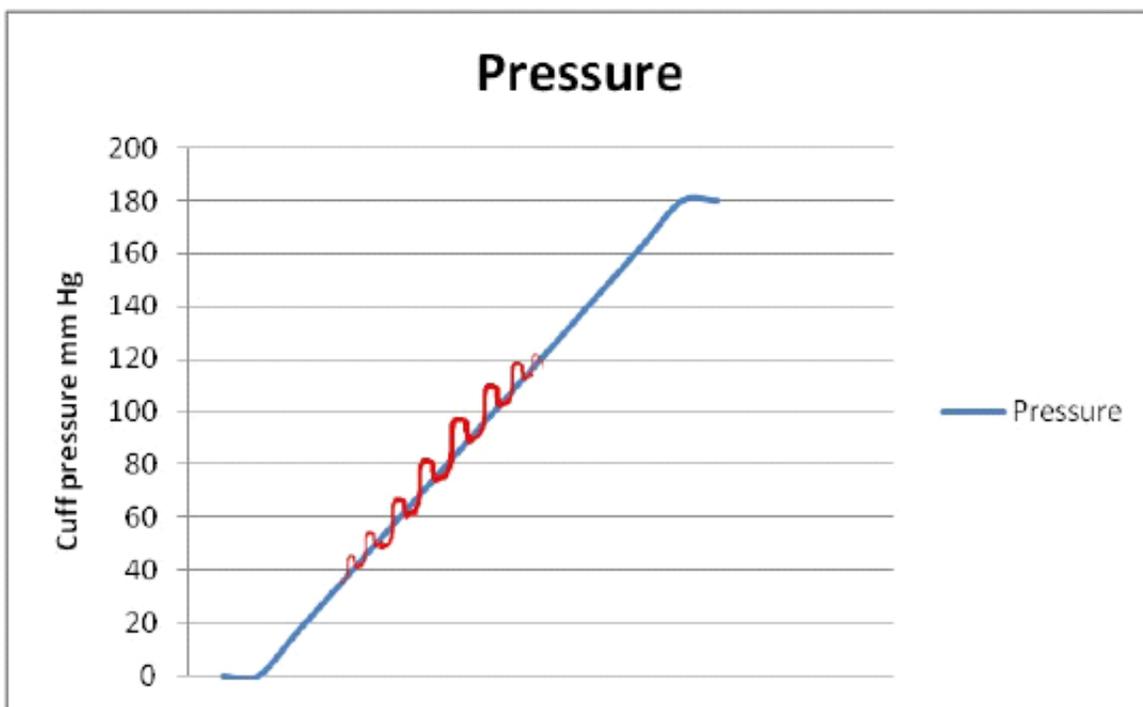


Figure 7. Pressure oscillations

These oscillations are constantly checked. On each oscillation, the main cuff pressure is measured and the oscillation amplitude is saved. When the pressure has reached a maximum value, the motor stops inflating and valve opens in order to deflate the cuff. While cuff is deflating calculations are performed.

### Software model

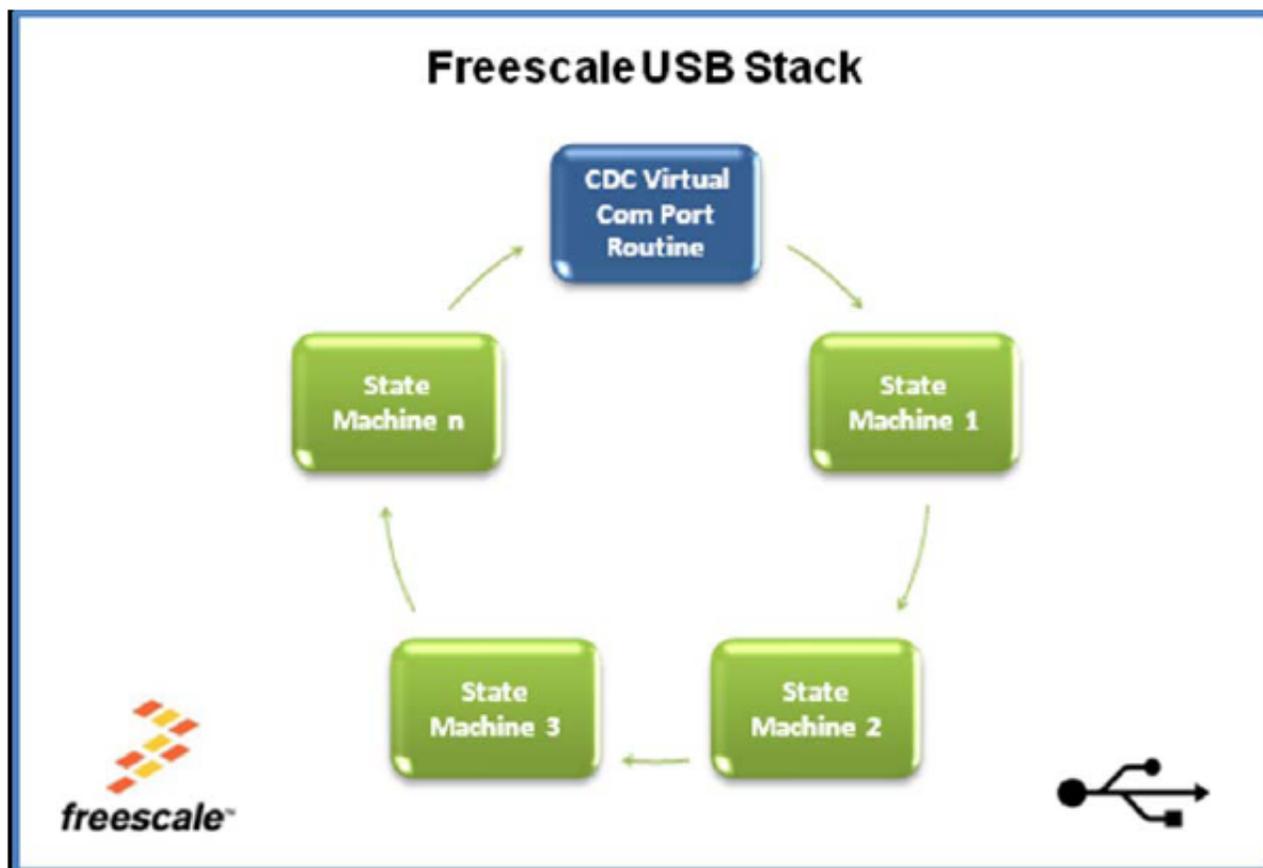
First, all the pulses are checked in order to find the one with the largest amplitude. The pulse with the largest amplitude represents the MAP, and the main cuff pressure recorded during this pulse is registered as MAP.

Systolic and diastolic arterial pressures are calculated using the method mentioned previously in [Blood pressure monitor operating principle](#).

## 4 Software model

The MED-BPM demo is based on the Freescale USB stack and behaves as a USB CDC (Communication Device Class). The demo works using state machines which execute one state per cycle, to avoid CPU kidnapping and emulating parallelism.

Figure 8 shows the general software model.



**Figure 8. General software model**

Each state machine is a task that has to be performed by the MCU. The system can perform several tasks, completing one at a time and not running the next one until the previous one is accomplished in a FIFO (First In First Out) order. Each state machine contains several sub-state machines allowing equal distribution of the CPU load among all the state machines. As mentioned before, software is based on the Freescale USB Stack with PHDC. More information about this software can be found in the USB Stack with PHDC API Reference Manual available at [freescale.com](http://freescale.com).

MED-BPM software is divided in three main parts—Initialization, communication with PC, and measurement execution.

## 4.1 Initialization

The first step when running MED-BPM demo is to initialize all the peripherals needed for demo execution. On the main() function, function Init\_Sys() is called first. This function initializes the clock and interruptions for working with USB. After this, some required peripherals for AFEs and the Software Timer are initialized for first use. USB is initialized as a CDC (Communication Device Class) so communications with the host can start. After this, the state machines execute in an infinite loop. The following figure depicts the initialization routine.

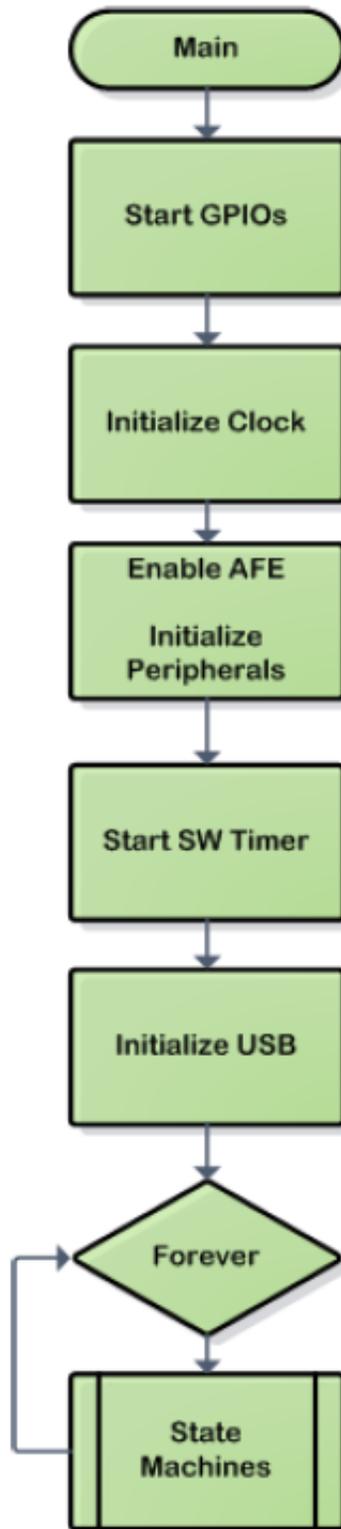


Figure 9. Initialization routine

## 4.2 Communication with PC

The PC communicates with the device via USB. The device is configured for working as a CDC (Communication Device Class) and behaves as a Virtual Com Port installed on the computer.

### 4.2.1 Command reception

Function `SerialComm_PeriodicTask` is the CDC Virtual Com Port routine and is called by the main program. This function is constantly checking the USB input buffer for received data. When a data packet has been received, the function checks if the received packet is a request according to the communication protocol. If it is, the function checks the command requested and executes it. [Figure 10](#) shows the flow diagram of the function `SerialComm_PeriodicTask`.

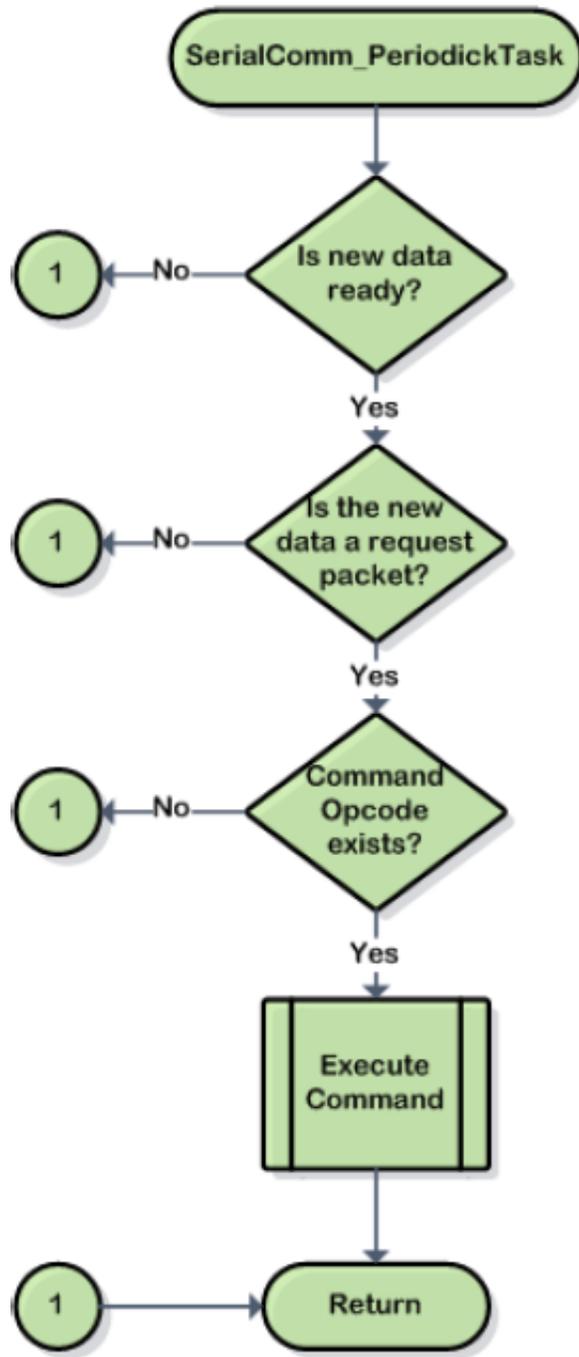


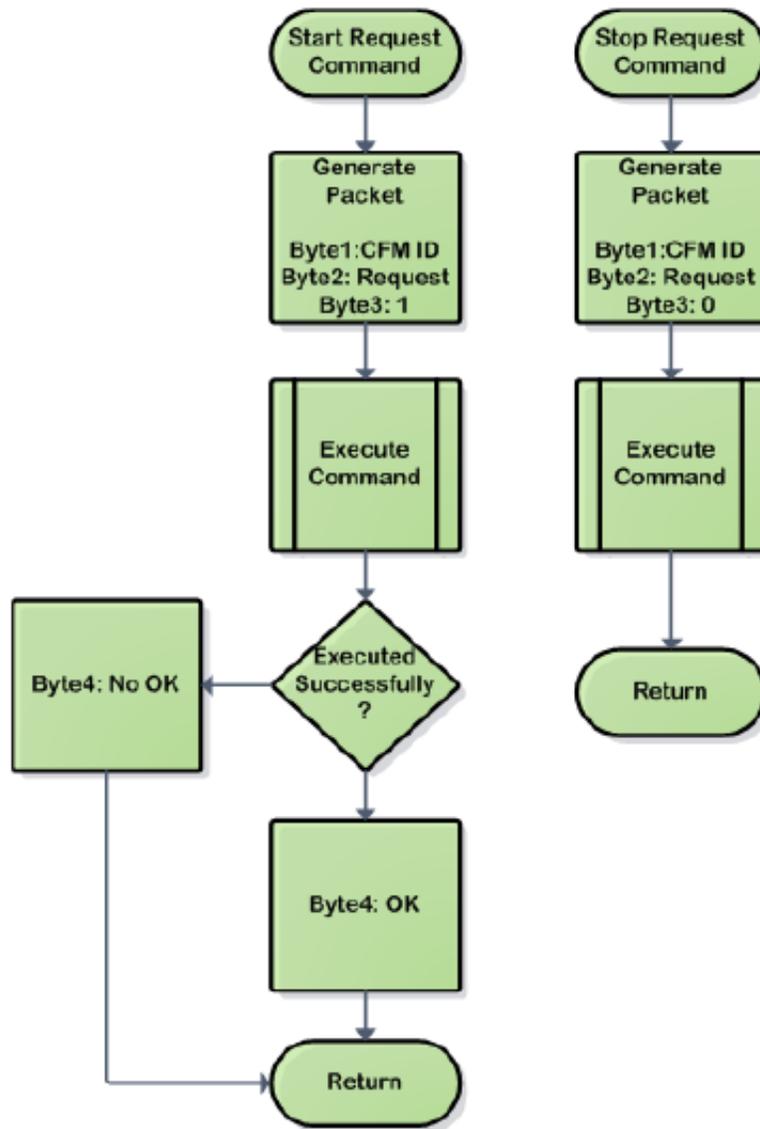
Figure 10. SerialComm\_PeriodickTask flow diagram

### 4.2.2 Command execution

MED-BPM recognizes the four request commands.

- BpmStartMeasurementReq: Starts blood pressure measurements.
- BpmStopMeasurementReq: Stops blood pressure measurements.
- BpmStartLeakTestReq: Starts leakage tests on the cuff.
- BpmStopLeakTestReq: Stops leakage tests on the cuff.

When any of these commands are executed, a confirmation packet is generated according to the Communication Protocol (see [Communication protocol](#)) indicating that the command has been received. When Start Requests are executed, a confirmation packet also indicates if the command was successfully executed or not. [Figure 11](#) shows a request command flow.



**Figure 11. Request command flow diagram**

### 4.2.3 Sending packets

Function SerialComm\_SendData sends the data packets to host. Data packets are stored on the output buffer when they are created and a data counter variable increases indicating the size of the output buffer. When the function SerialComm\_SendData is called, it checks the size of the data counter variable. If it is not zero, it means that there is information in the output buffer that needs to be sent. The function calls the CDC interface, part of the USB Stack with PHDC to send the packet. [Figure 12](#) shows the function’s flow diagram.

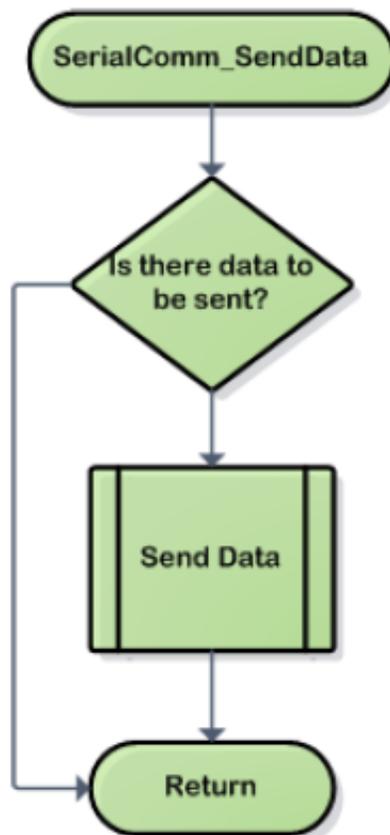


Figure 12. SerialComm\_SendData flow diagram

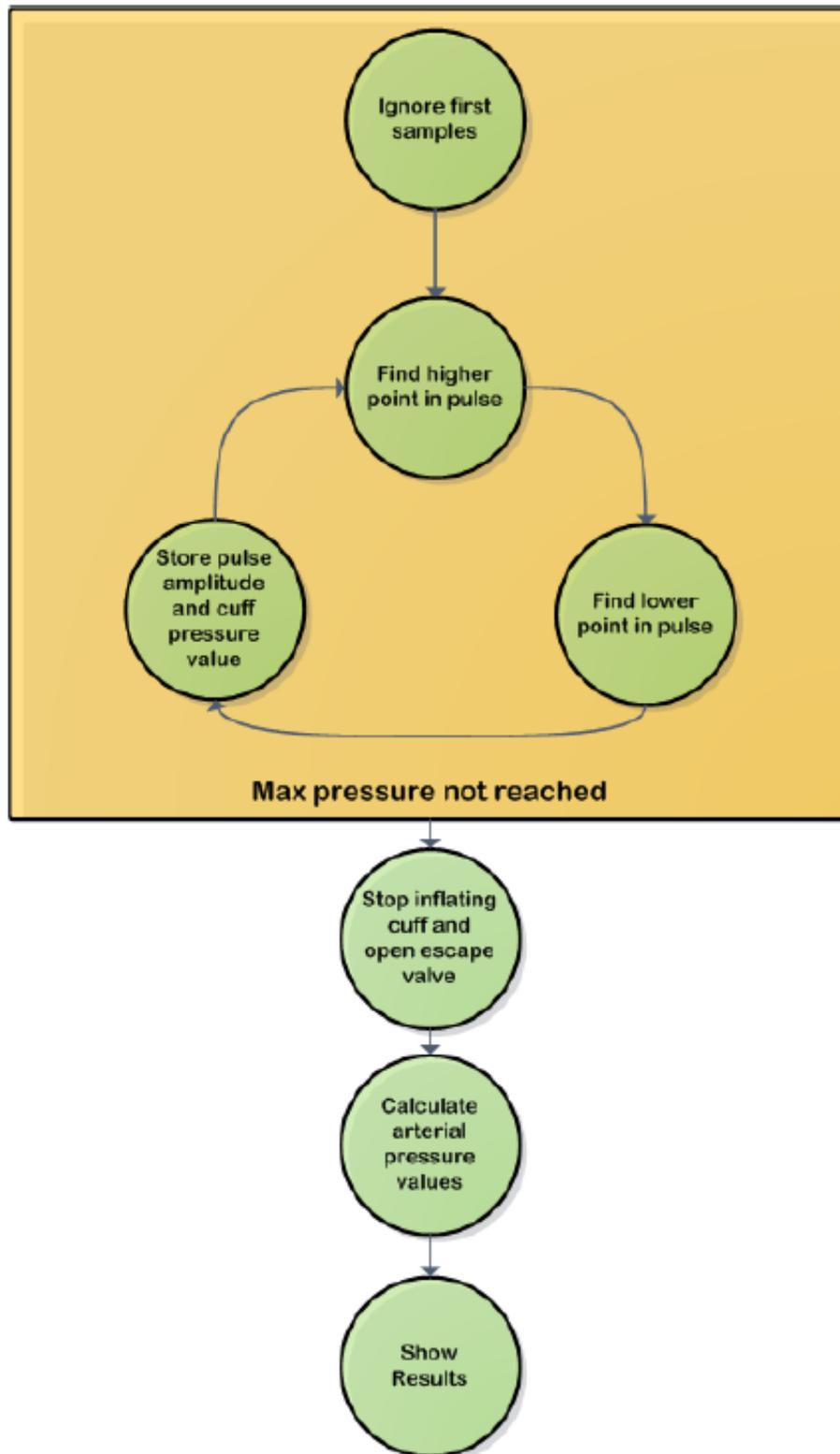
## 4.3 Measurement execution

When the function `BpmStartMeasurementReq` is executed, the function `Bpm_StartMeasurement` is called. This function initializes BPM so measurements can be performed. This function first resets all the variables to ensure the initial state, then it initializes ADCs for working with a 12-bit resolution. Because the first ADC measurements do not represent any useful information for arterial pressure calculation, those samples are ignored. `BpmIgnoreSamplesCounter` is loaded with a predetermined quantity of samples to be ignored.

`BpmActualState` is set to `STATE_MEASURING` indicating to the BPM state machine that measurements have to be performed. A Software Timer is now started for taking an ADC sample each 10 ms. More information about the Software Timer can be found in [Software timer](#). Finally, the escape valve is closed and air pump motor is activated to start measurements.

### 4.3.1 State measuring

In the function `Bpm_StartMeasurement`, the BPM state machine is taken out of its idle state and set to State Measuring. On the next state machine execution, the function `StateMeasuring` is called. [Figure 13](#) shows the `StateMeasuring` flow diagram.



**Figure 13. StateMeasuring flow diagram**

During the ramp-up period, cuff pressure is being constantly checked and compared with a maximum reference value in order to avoid damages to the patient from an excessive inflation. Because the first samples under 40 mmHg do not represent any useful information to be analyzed, those samples are ignored during the first sub-state. A sample counter is previously defined indicating the quantity of samples to be ignored.

## Running blood pressure monitor demo

After first samples are ignored, the program searches for oscillations on the main cuff pressure. It looks for the higher point in each pulse by comparing the new sample with the previous one. If the new sample is bigger than the previous one, the new sample is set as the higher point in the actual pulse until a higher sample appears. If, after five samples, the new sample is smaller than the previous one, it is considered that the pulse is now going down and the process of looking for the lowest point on the actual pulse starts. After both pulses have been determined, the pulse amplitude is calculated and stored into an array with the main cuff pressure measured in that moment, in order to be analyzed later.

When the maximum allowable pressure in the cuff is reached, the air pump motor is stopped and escape valve is opened. While cuff is being deflated, calculations are performed into the MCU. The first step is to determine the MAP by finding the pulse with the biggest amplitude previously stored in the array. The main cuff pressure measured during this pulse is considered to be the MAP. After this, systolic arterial pressure is determined by finding a pulse with a main cuff pressure above the MAP and with amplitude of a 70% of the MAP pulse. Diastolic pressure is determined by finding a pulse with a main cuff pressure under the MAP and with amplitude of a 50% of the MAP pulse. Finally, ADC values are converted into mmHg in order to be sent to the GUI to be displayed.

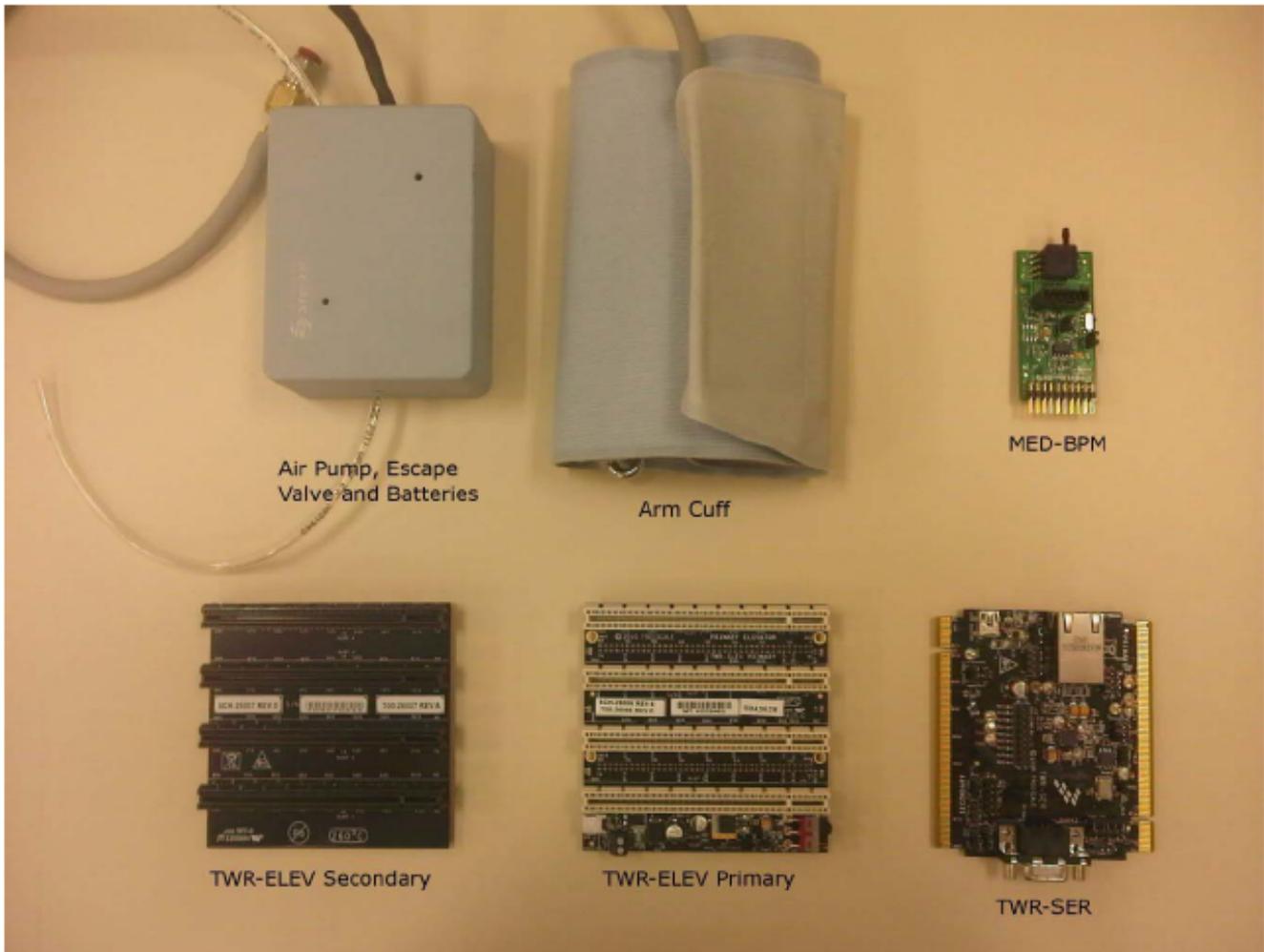
## 5 Running blood pressure monitor demo

The Blood Pressure Monitor Demo was developed for running on the Freescale Tower System. The following steps will guide users on how to run the Blood Pressure Monitor Demo.

### 5.1 Tower system configuration

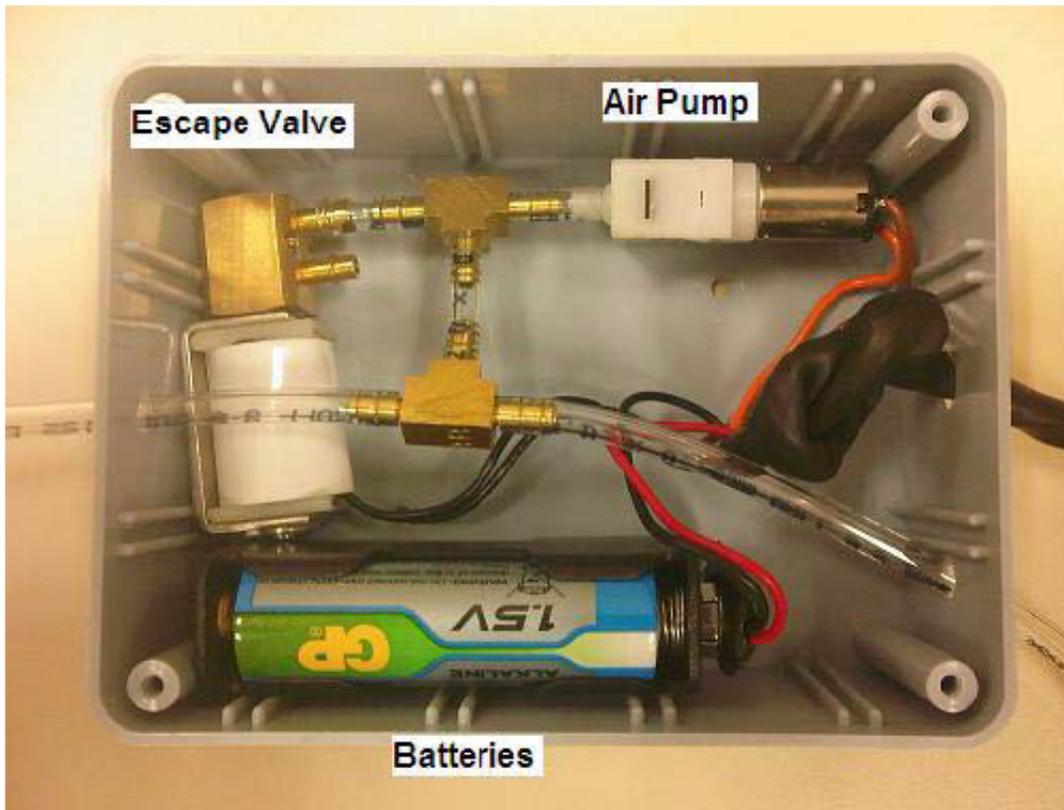
The Freescale Tower System is a modular development platform which allows rapid prototyping, a shorter time to market, and tool reuse. The Blood Pressure Monitor Demo requires five boards to work, 2 x TWR-ELEV boards, 1 x TWR-SER, 1 x MED-BPM AFE and a TWR Controller module which can be TWR-K53N512, TWR-MCF51MM or a TWR-S08MM128. MED-BPM AFE can only be connected to microcontroller modules that include a medical connector. The other Tower modules can be used together with other modules. More information about the Freescale Tower System can be found on [freescale.com/tower](http://freescale.com/tower).

The following image shows the elements needed to run the Blood Pressure Monitor demo.



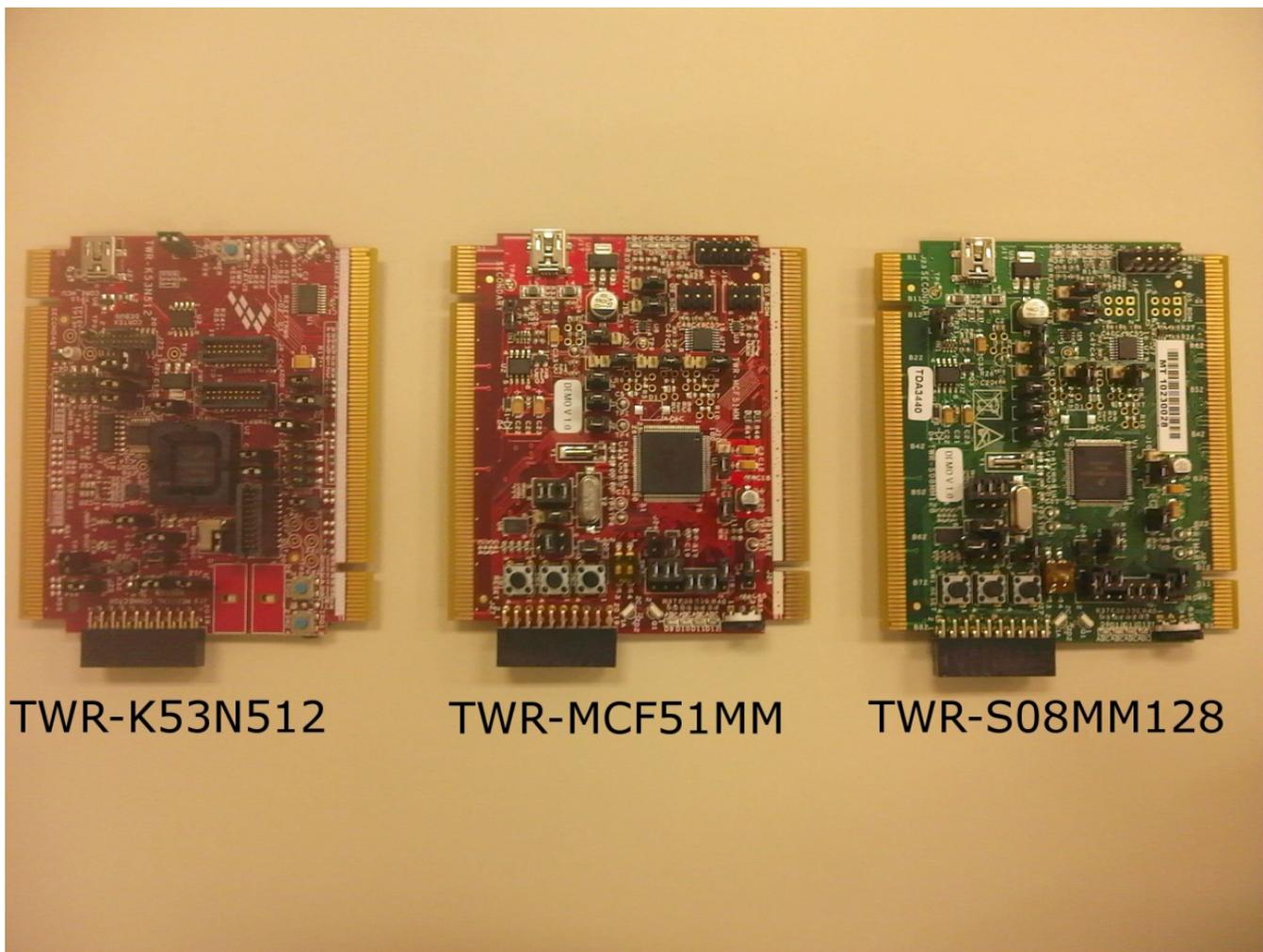
**Figure 14. Elements required to run BPM demo**

The blue box contains the air pump, the escape valve, and 2 X AA batteries for driving motor and valve because the USB port cannot provide sufficient current for those devices. [Figure 15](#) shows the elements in the blue box.



**Figure 15. External control elements**

The Blood Pressure Monitor demo works using TWR-K53N51, TWR-MCF51MM, and TWR-S08MM128 controller modules. The following image shows the three controller boards.



**Figure 16. Controller boards**

### 5.1.1 Jumper configuration

To run the blood pressure monitor properly, the following jumper configurations must be made on the boards.

TWR-SER jumper configuration

- J10 → 1-2
- J16 → 3-4

TWR-K53N512 jumper configuration

- J1 → Open
- J3 → Open
- J4 → 1-2
- J11 → 1-2
- J15 → 1-2
- J17 → 1-2
- J18 → 1-2

TWR-MCF51MM and TWR-9S08MM128 jumper configuration

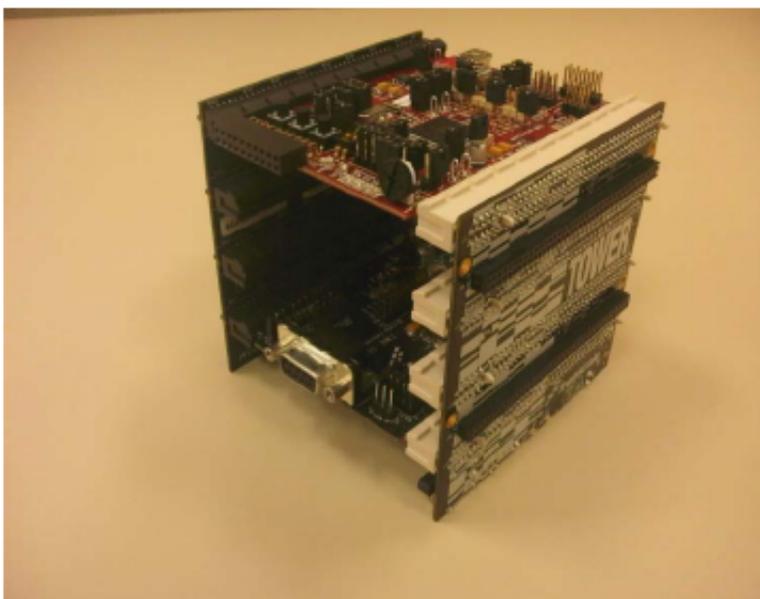
- J1 → 1-2
- J2 → 1-2

## Running blood pressure monitor demo

- J10 → 2-3
- J11 → 1-2
- J12 → Open
- J18 → 1-2, 9-10, 11-12 and 13-14 Open

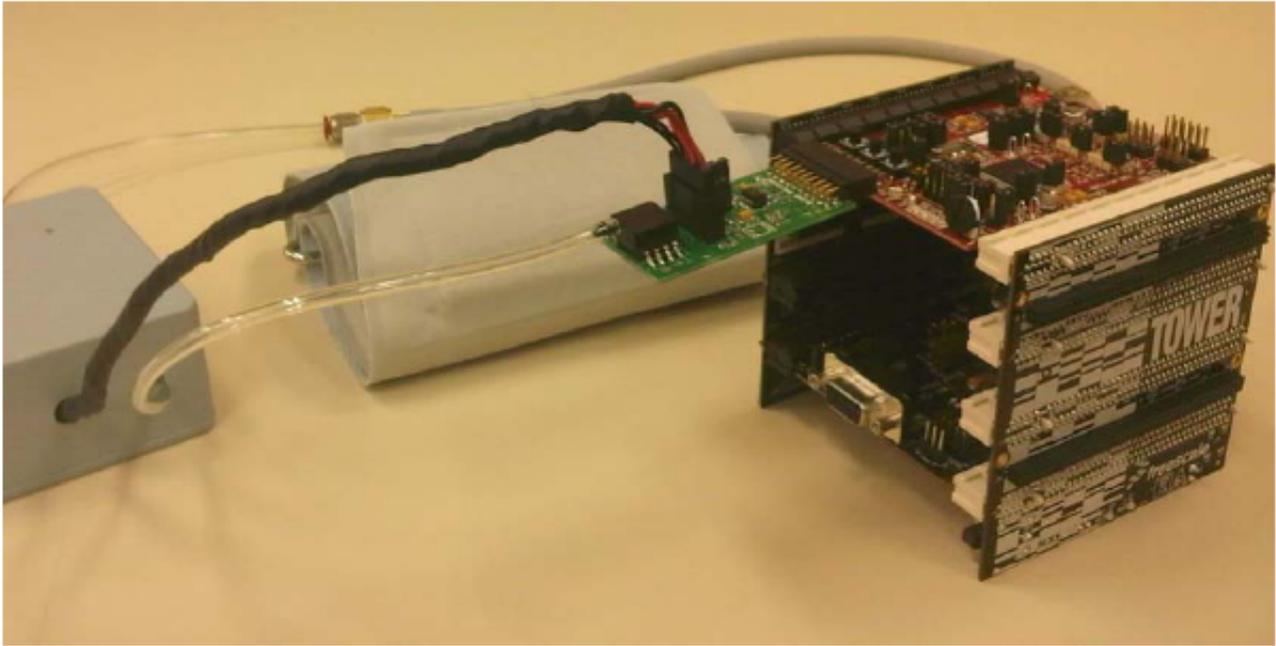
### 5.1.2 System assembly

The Tower system must be assembled by connecting the controller module and the TWR-SER board to the TWR-ELEV boards. The side of the Controller and TWR-SER boards marked as “Primary” must be connected to the Primary Elevator and the other side to the Secondary Elevator. The MED-BPM board must be connected to the Controller Board’s medical connector. The following image shows the tower system assembled.



**Figure 17. Tower system assembled**

Connect the external components to the MED-BPM so they can be controlled by the MCU. Batteries, Air Pump motor and valve cables must be connected to the MED-BPM board as shown in [Figure 5](#) on the external elements connector. A single air tube is connected to the cuff, the air pump, and the escape valve. This tube must be connected to the pressure sensor in order to be constantly measuring the pressure values on the system. Once the system has been completely assembled, it will look like [Figure 18](#).



**Figure 18. Blood Pressure Monitor demo assembled**

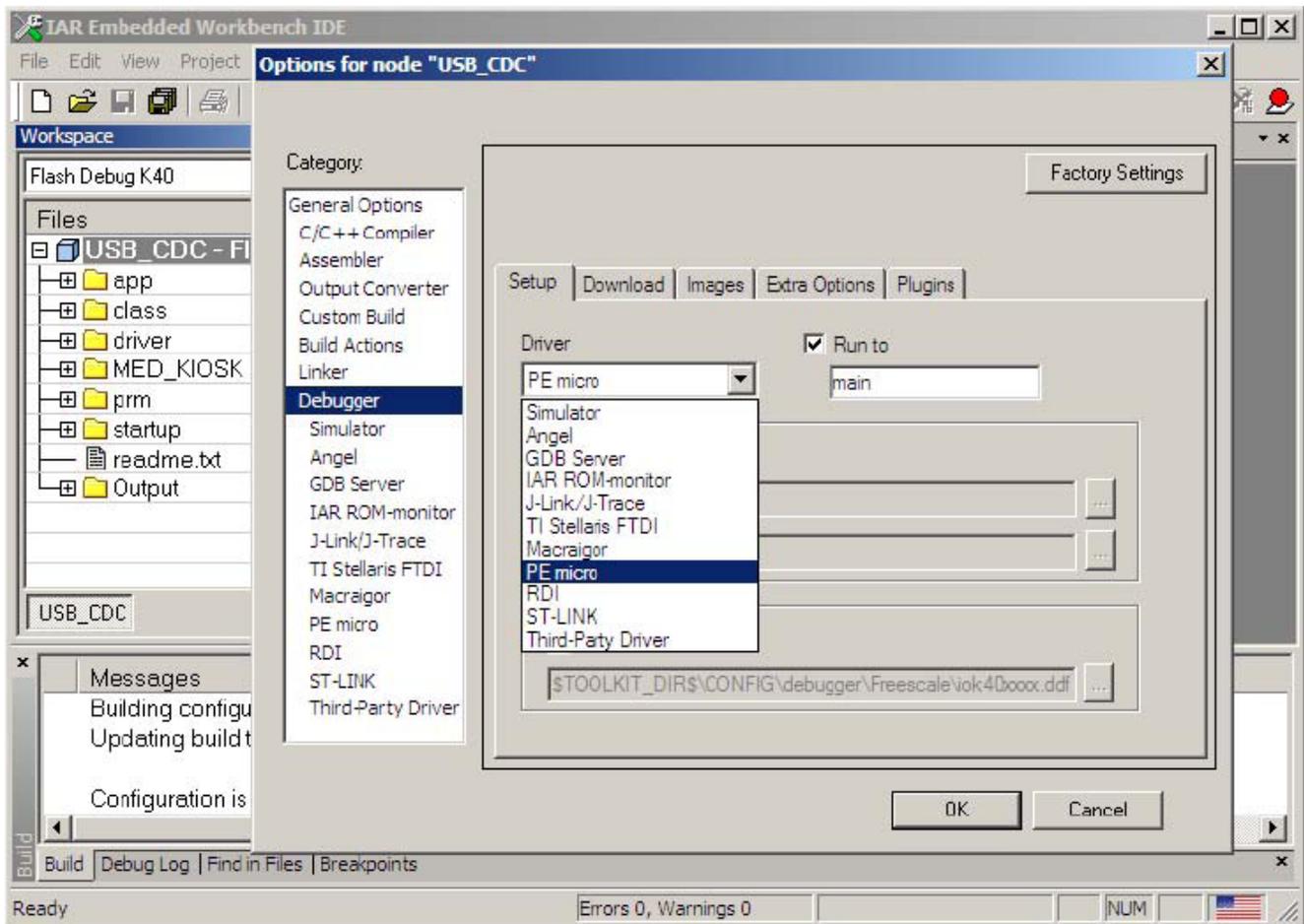
## 5.2 Loading application

Software for Blood Pressure Monitor demo was developed on IAR Workbench for ARM<sup>®</sup> 6.10 for the Kinetis K53 MCU and in Freescale CodeWarrior 6.3 for MCF51MM256 and MC9S08MM128 MCUs. Each one of these compilers loads the program into the MCU in a different manner. This part of the document will explain how to load the program on each type of compiler.

### 5.2.1 Loading IAR project on K53

The following steps will guide the user to load the IAR project on K53.

1. Connect the USB cable to the TWR-K53N512 board. The device must be recognized as an Open Source BDM–Debug Port. Install the Open Source BDM drivers in the IAR folder.
2. Choose Menu > Project > Options to open the project and verify the debugger in the project options panel. Select the category Debugger and check that PE micro is selected ([Figure 19](#)).



**Figure 19. Debugger selection**

3. Load the project into the MCU by clicking the DEBUG button (Figure 20). The project will compile automatically and load into the MCU. After that, disconnect the USB cable from the TWR-K53N512 board.

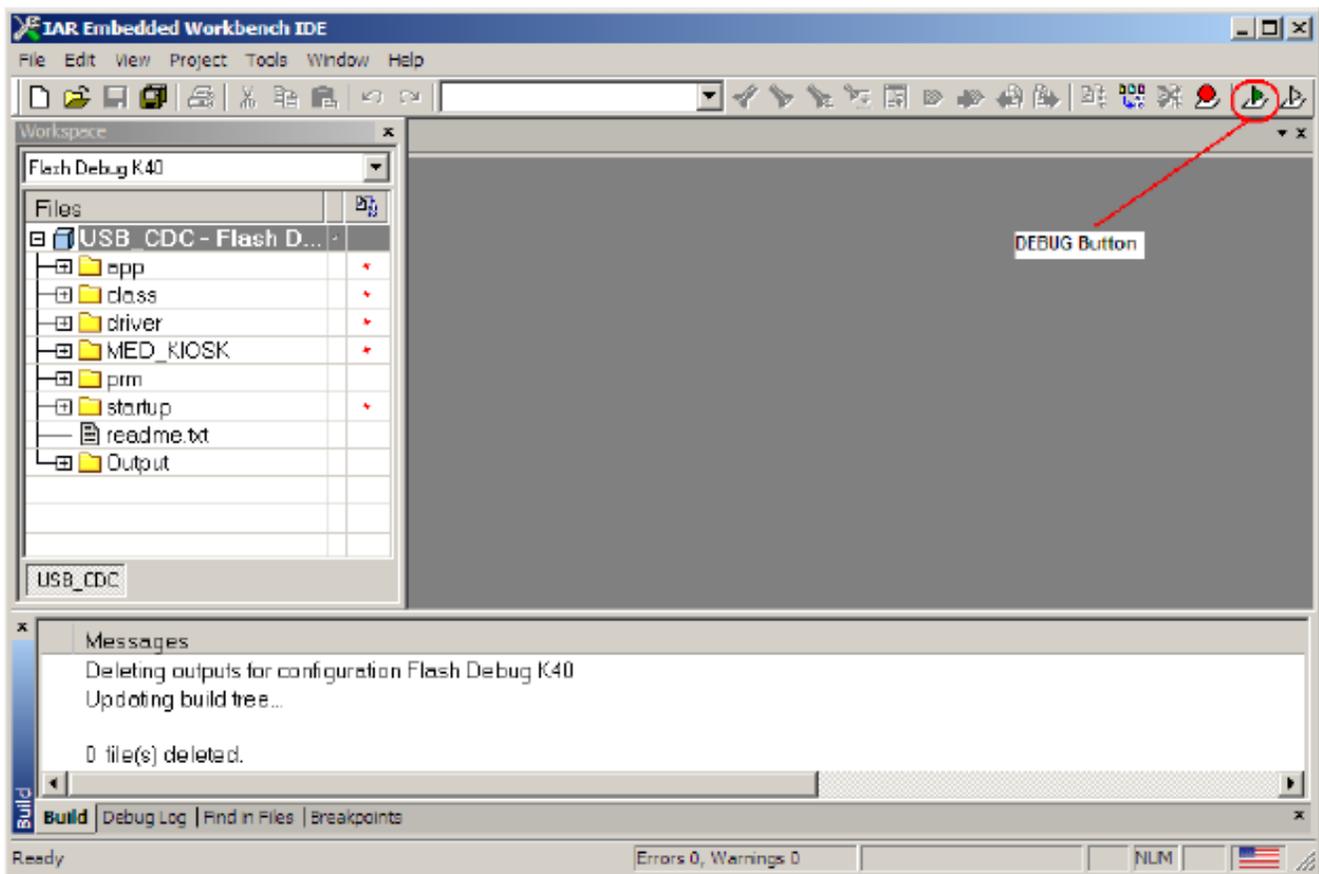
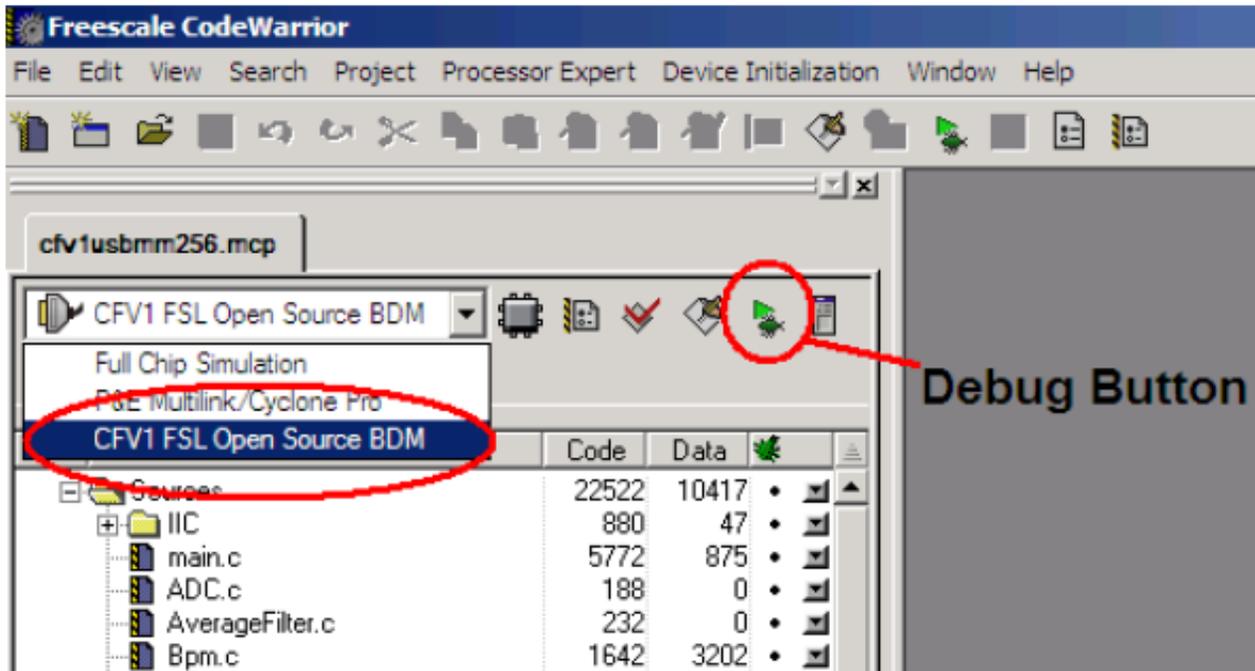


Figure 20. Debug button

## 5.2.2 Loading CodeWarrior project on MM devices

The following steps guide the user to load the CodeWarrior project on MM devices.

1. Connect a USB cable to the TWR-SER board in order to turn on the board. Connect a USB cable to the TWR-MM Controller module. The device must be recognized as an Open Source BDM–Debug Port. Install the Open Source BDM drivers on the CodeWarrior folder.
2. Open the CodeWarrior project for the MM module to be used and verify that the option Open Source BDM is selected for debugging (Figure 21). Then, press the Debug button to automatically compile the program and load it into the MCU.



**Figure 21. CodeWarrior window**

3. Once the program has been loaded into the MCU, disconnect the USB cable from both TWR-SER and MM controller boards.

### 5.3 Running demo

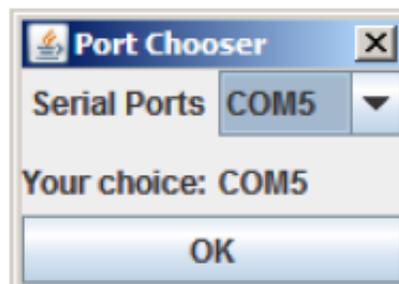
Once the Tower System has been configured and the project loaded into the MCU (Tower system configuration & Loading application), demo can be used. The following steps must be completed in order to run the demo successfully.

1. Connect the USB cable to the TWR-SER board as shown in [Figure 22](#).



**Figure 22. Connecting TWR-SER USB cable**

2. The device must be recognized as a Virtual Com Port. Install the driver for Virtual Com Port included in the project folder.
3. Once the driver has been installed and device recognized, open the Graphic User Interface (GUI). A window will appear asking for the COM PORT number assigned to the device. Select the appropriate COM PORT number and click OK (Figure 23). The GUI interface will appear on screen. (Figure 24).

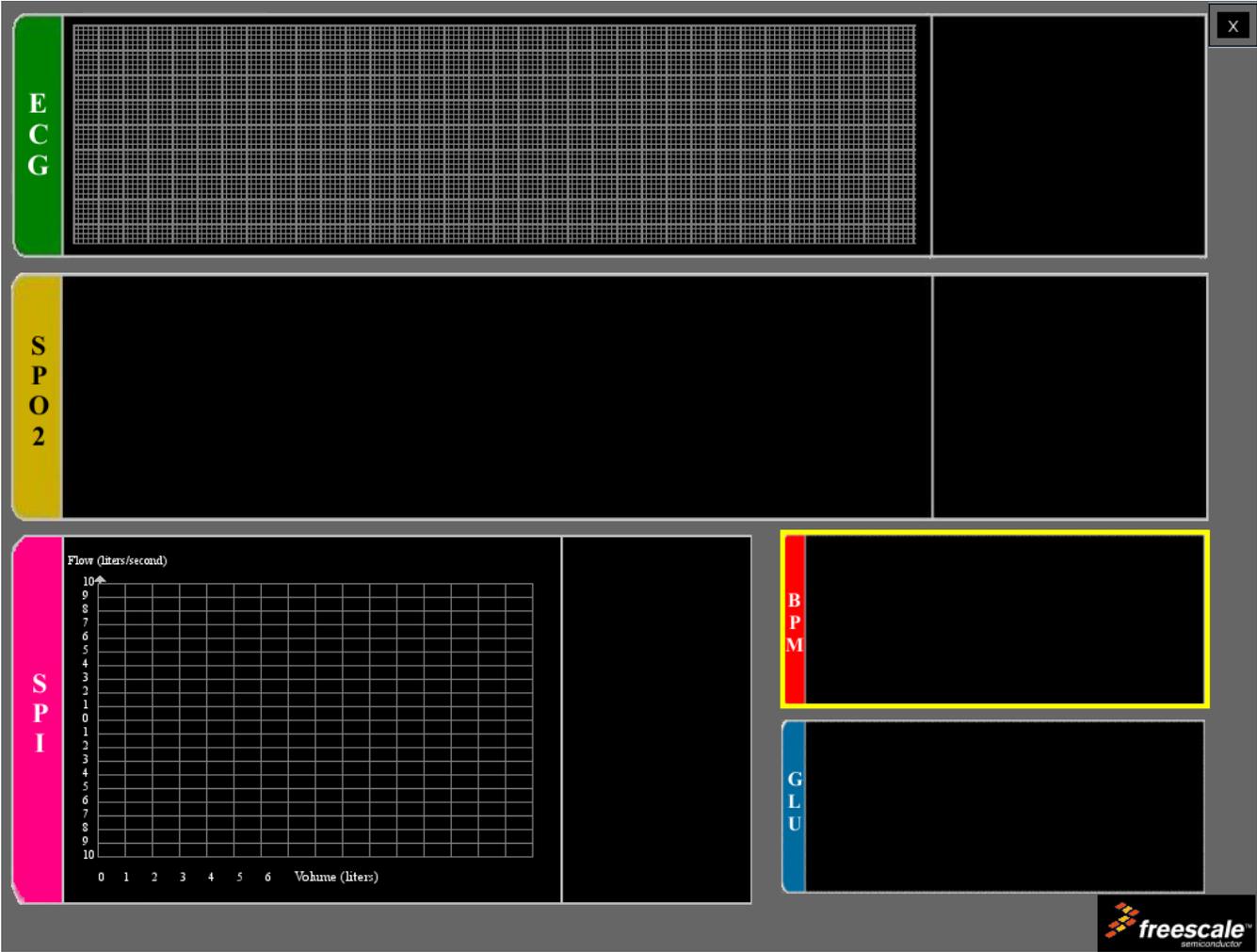


**Figure 23. Port chooser**



**Figure 24. GUI main screen**

4. Verify that Caps Lock is not activated on keyboard and then press Shift + D in order to start GUI in doctor mode (Figure 25).



**Figure 25. Doctor mode**

- 5. Place the cuff on the left arm with the plastic hose pointing to the brachial artery, and adjust the cuff to the arm size as shown in [Figure 26](#).



**Figure 26. Cuff placement**

6. Click the BPM area on the GUI. Measurements will start and after a while, the measurements will appear on screen ([Figure 27](#)).

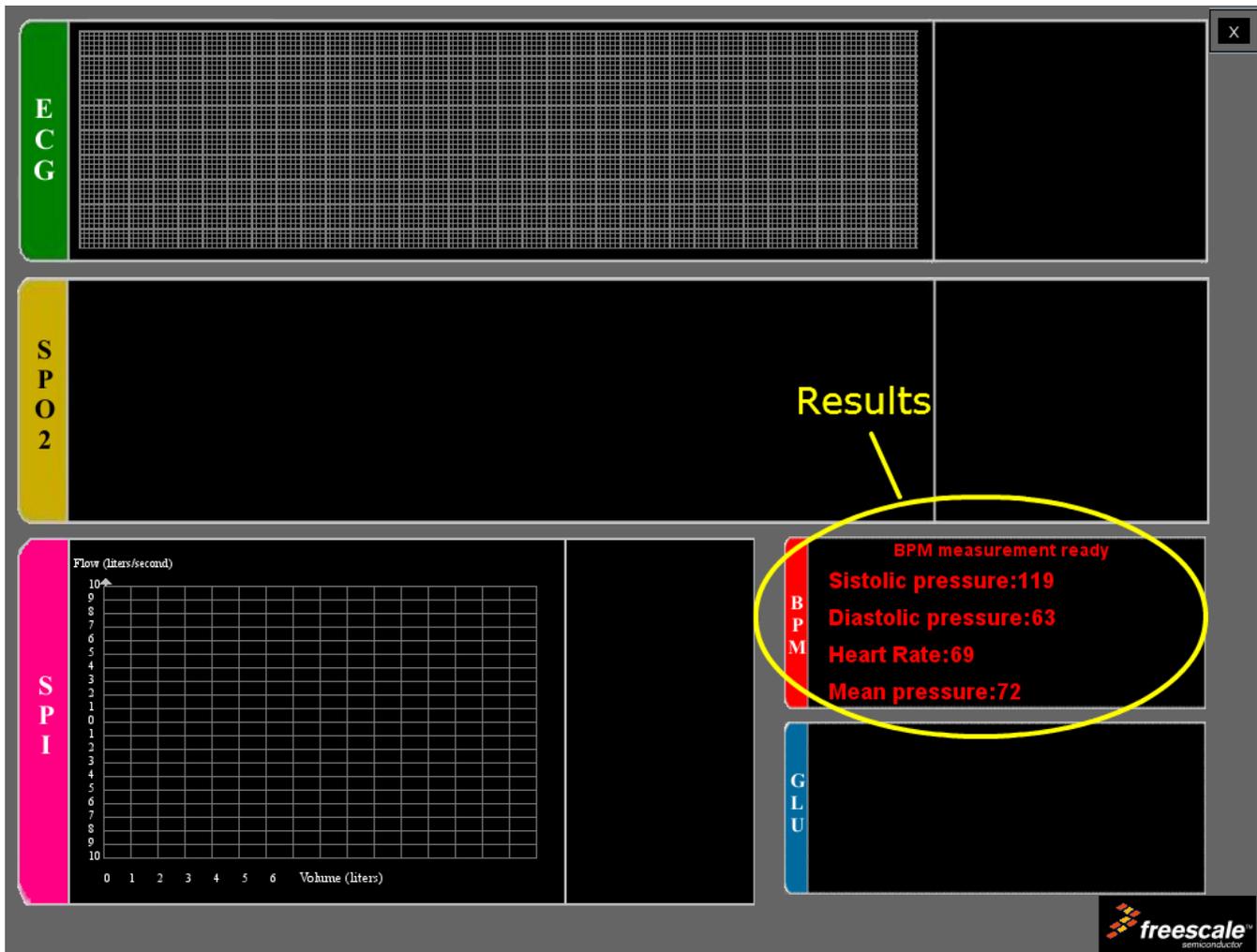


Figure 27. GUI running

## 6 References

Following is a list of online resources related to this application note, available on the Freescale website.

- More information about Freescale Medical portfolio can be found on [freescale.com/Medical](http://freescale.com/Medical).
- More Tower System information can be found on [freescale.com/tower](http://freescale.com/tower).
- Software for MM devices was developed using CodeWarrior V6.3 and can be found on [freescale.com/CodeWarrior](http://freescale.com/CodeWarrior).
- Software for Kinetis was developed and tested on IAR v6.1 for ARM. Download 30-day evaluation software from IAR Webpage in the section Downloads. ([iar.com/Downloads](http://iar.com/Downloads))
- Follow us at [facebook.com/freescale](https://facebook.com/freescale), [twitter.com/freescale](https://twitter.com/freescale), and [youtube.com/freescale](https://youtube.com/freescale).

## 7 Conclusions

Freescale Medical portfolio offers a wide range of solutions for medical developments. Kinetis K50 family and Flexis MM are examples of high-performance, ultra low-power MCUs embedding analog peripherals, ideal for biosignals treatment and great processing capabilities for digital signal treatment.

The Blood Pressure Monitor is an example of the capabilities of the Freescale Medical MCU portfolio, allowing the development of medical applications using 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.

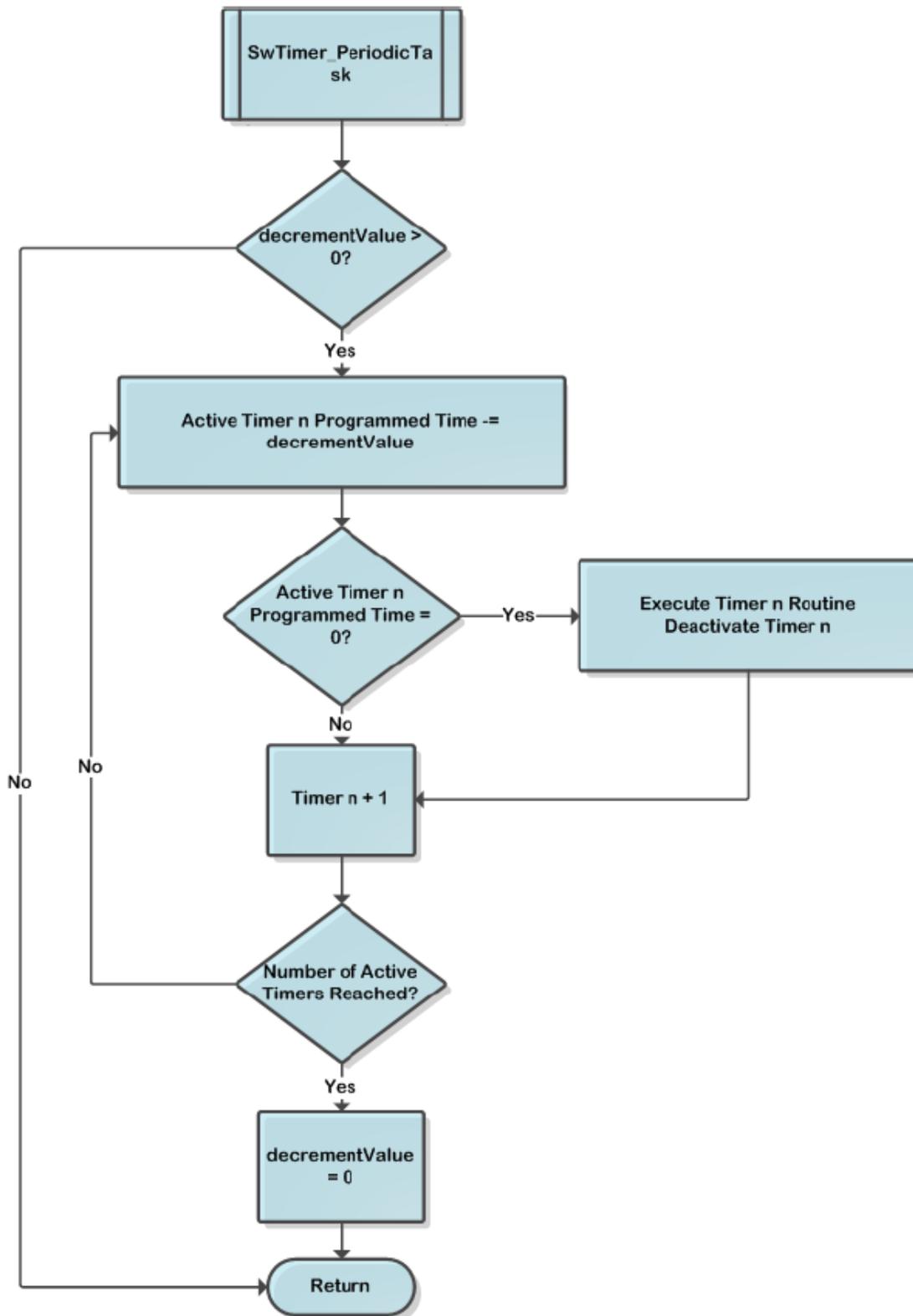


Figure A-1. Block diagram

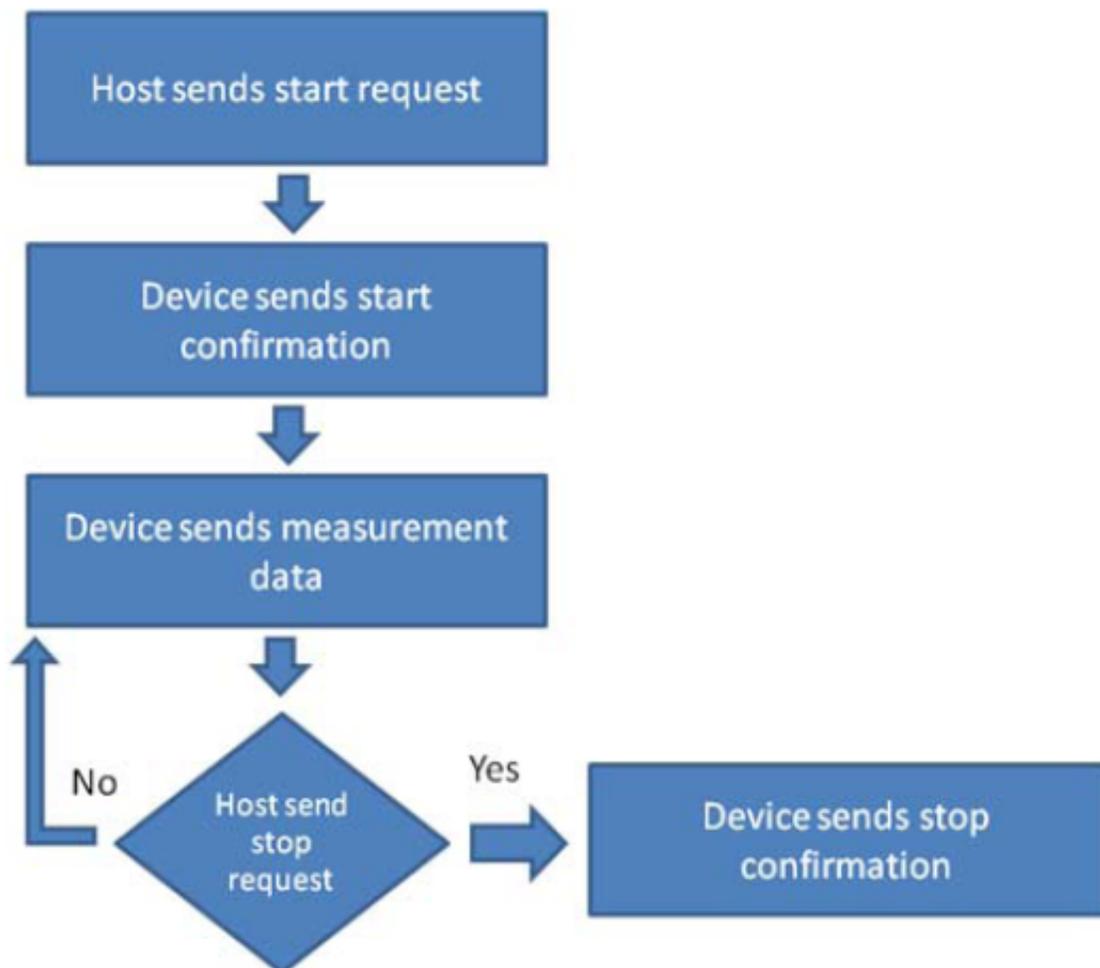
## Appendix B Communication protocol

The application communicates with the Graphic User Interface (GUI) 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

## Packet type

- 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 to be 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<sup>1</sup>

**Table B-2. Opcodes**

Opcode	REQ	CFM	IND	Opcodes (Hex)
<b>Glucose meter</b>				
GLU_START_MEASUREMENT	X	X		0x00
GLU_ABORT_MEASUREMENT	X	X		0x01
GLU_START_CALIBRATION	X	X		0x02
GLU_BLOOD_DETECTED			X	0x03
GLU_MEASUREMENT_COMPLETE_OK			X	0x04
GLU_CALIBRATION_COMPLETE_OK			X	0x05
<b>Blood Pressure Meter</b>				
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
<b>Electro Cardiograph Opcode</b>				
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
<b>Thermometer</b>				
TMP_READ_TEMPERATURE	X	X		0x15
Height scale				
HGT_READ_HEIGHT	X	X		0x16
Weight scale				
WGT_READ_WEIGHT	X	X		0x17
<b>Spirometer</b>				
SPR_DIAGNOSTIC_MODE_START_MEASUREMENT	X	X		0x0C

*Table continues on the next page...*

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

**Table B-2. Opcodes (continued)**

Opcode	REQ	CFM	IND	Opcodes (Hex)
SPR_DIAGNOSTIC_MODE_STOP_MEASUREMENT	X	X		0x0D
SPR_DIAGNOSTIC_MODE_NEW_DATA_READY			X	0x0E
<b>Pulse oximetry</b>				
SPO2_START_MEASUREMENT	X	X		0x21
SPO2_ABORT_MEASUREMENT	X	X		0x22
SPO2_MEASUREMENT_COMPLETE_OK			X	0x23
SPO2_MEASUREMENT_ERROR			X	0x24
SPO2_DIAGNOSTIC_MODE_START_MEASUREMENT	X	X		0x25
SPO2_DIAGNOSTIC_MODE_STOP_MEASUREMENT	X	X		0x26
SPO2_DIAGNOSTIC_MODE_NEW_DATA_READY			X	0x27
				0x21
<b>System commands</b>				
SYS_CHECK_DEVICE_CONNECTION	X	X		0x29
SYS_RESTART_SYSTEM	X			0x2A

### 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**

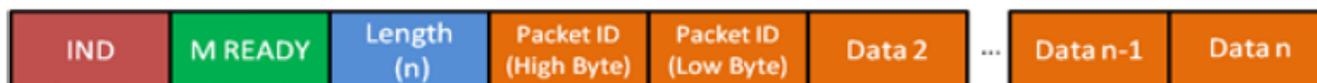
The error byte indicates the device status. The table below shows possible error codes.

**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](http://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](http://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](http://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](mailto: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](mailto:support.asia@freescale.com)

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductors products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claims alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-complaint and/or non-Pb-free counterparts. For further information, see <http://www.freescale.com> or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to <http://www.freescale.com/epp>.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© 2012 Freescale Semiconductor, Inc.