Freescale Semiconductor
Application Note

Document Number:AN4364
Rev. 1, 01/2013

Glucose Meter Fundamentals and Design

by: Miriam Garcia Yanez

Contents

1 Introduction

This application note shows a basic glucometer using Freescale products to determine approximate concentration of glucose in blood.

This glucometer can be implemented with K53 microcontroller of the Kinetis family and with MCF51MM and S08MM128 MCUs, part of the Flexis MM family. It includes Freescale USB stack to show data through a graphic user interface (GUI) in a PC.

This application note is for anyone with an interest in glucometers as a medical electronic application, such as biomedical engineers, physicians, medical equipment developers, or someone out of those fields but with an interest in the operation of a glucometer.

2 Glucometer fundamentals

This section provides basic explanation of blood glucose regulation process in a human body and principle functionality of a glucometer.

2.1 Natural blood glucose regulation

Glucose ($C_6H_{12}O_6$) is a carbohydrate whose most important function is to act as a source of energy for the human body, by being the essential precursor in the synthesis of ATP (adenosine triphosphate). The energy stored in ATP can then be used to drive processes requiring energy, including biosynthesis, and locomotion or transportation of molecules across cell membranes. According to cellular requirements, glucose can also be used in the creation of proteins, glycogen, and lipids.

The blood glucose concentration is very tightly regulated. Human body has two hormones released by pancreas that have opposite effects: insulin and glucagon. Insulin is produced by beta cells of the pancreas while glucagon is produced by alpha cells. The release of insulin is triggered when high levels of glucose are found in the bloodstream, and glucagon is released with low levels of glucose in the blood.

This blood glucose regulation process can be explained in the following steps:

1. After the glucose has been absorbed from the food eaten, it gets released in the bloodstream. High blood glucose levels triggers the pancreas to produce insulin. Insulin enables the muscle cells to take glucose as their source of energy and to form a type of molecule called glycogen that works as secondary energy storage in the case of low levels of glucose. In the liver cells, insulin instigates the conversion of glucose into glycogen and fat. In the fat cells of the adipose tissue, insulin also promotes the conversion of glucose into more fat and the uptake of glucose.
2. The pancreas will continue to release insulin and liver and fat cells continue to use glucose till the drop of concentration of glucose is below a threshold; in that case, glucagon will be released instead of insulin.
3. When glucagon reaches the liver cells, it initiates the conversion of glycogen into glucose, and fat into fatty acids, which many body cells can use as energy after the glucagon enables them to. The cells will continue to burn fat from the adipose tissue as an energy source, and follow with the protein of the muscles, until the levels of glucose increase again by the digestion of food, and that terminates the cycle.

2.2 Diabetes mellitus

Diabetes is a chronic disease characterized by high or low blood glucose levels, which results from the pancreas not working properly and not producing enough insulin or when the body cells do not respond to it in the correct way. There are three types of diabetes:

1. Type 1 diabetes is also known as juvenile diabetes because it is typically diagnosed in children and young adults. In this type of diabetes, the body does not produce insulin. 5% of the population with diabetes has this type of illness.
2. Type 2 diabetes is the result of the body not producing enough insulin or the cells not using insulin properly. This is the most common form of diabetes. 90% of the population with diabetes has this type. Some of the risk factors are physical inactivity, excess body weight, genetics, age greater than 45 years, and ethnicity.
3. Gestational diabetes is high blood glucose levels first diagnosed during pregnancy. This does not mean that the woman will have diabetes after she gives birth or that she had it before she conceived, but it is a risk factor for type 2 diabetes in the future.

3 Glucometer implementation

This glucometer (referred as MED-GLU) is implemented using a Freescale MCU Kinetis K53, a MCF51MM or a S08MM128 MCU part of the Flexis MM family.

The features of Kinetis K53 and some of the peripherals in its integrated measurement engine are:

- Ultra low-power operation
- Two operational amplifiers (OPAMP)
- Two trans-impedance amplifiers (TRIAMP)
- 2 × 12-bit digital-to-analog converter (DAC)
- 2 × 16-bit SAR analog-to-digital converter (ADC), up to 31 channels with programmable gain amplifiers (PGA)
- Inter-integrated circuit (I2C)

- USB connectivity
- ARM® Cortex™ M4 core with digital signal processor (DSP) instructions

The MCF51MM and S08MM128 provide the following useful features and peripherals:

- Ultra low-power operation
- Two operational amplifiers (OPAMP)
- Two transimpedance amplifiers (TRIAMP)
- 16-bit SAR analog-to-digital converter (ADC), 4 differential channels and up to 12 external single-ended channels.
- 12-bit digital-to-analog converter (DAC)
- Inter-integrated circuit (I2C)
- Universal serial bus (USB) connectivity
- Multiply-accumulate unit (MAC only in MCF51MM)
- ColdFire V1 and HCS08 cores, respectively

The implementation still requires some external components. These components are integrated in an external Analog Front End (AFE) which is described below.

3.1 MED-GLU AFE

MED-GLU AFE includes all the external components necessary, except test trips, to implement a glucometer along with the Kinetis K53 MCU. The AFE functional block diagram is shown and described in figure below.

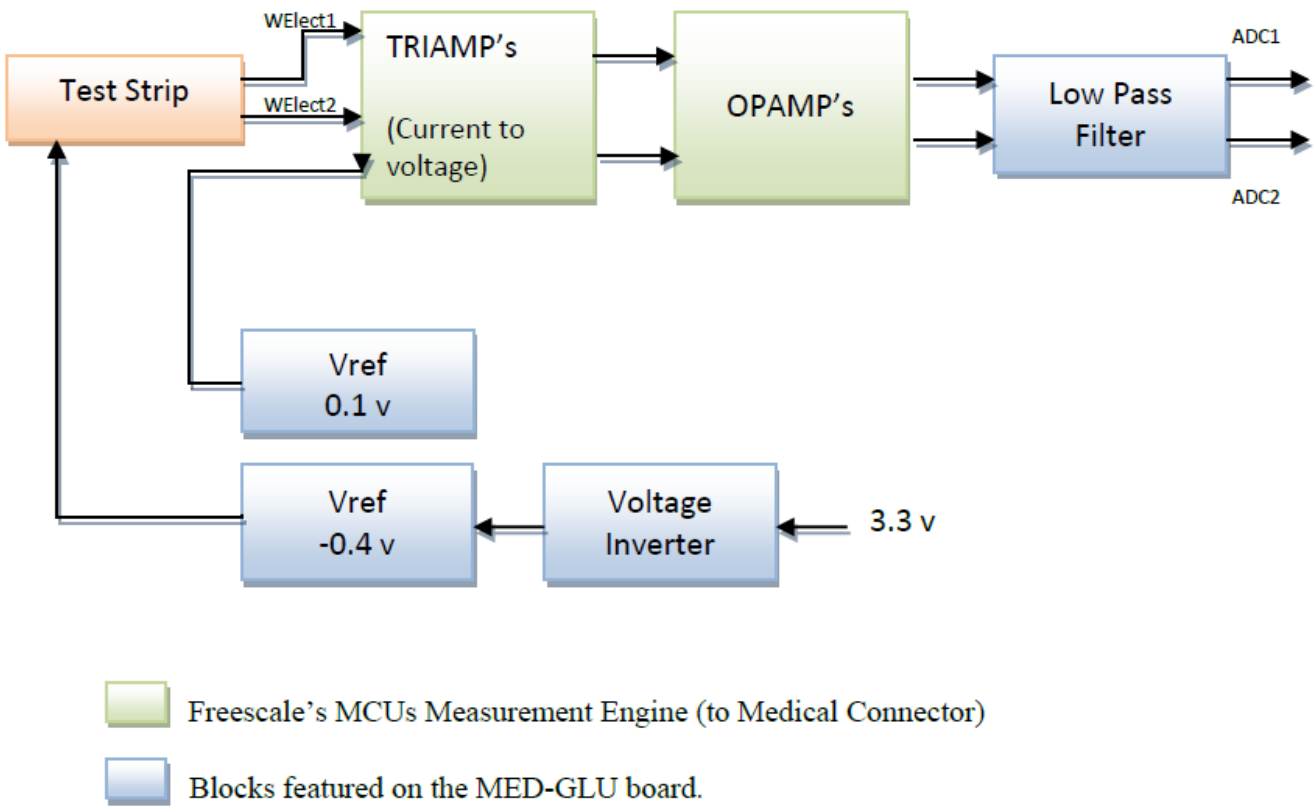

Figure 1. MED-GLU AFE functional block diagram

3.1.1 Medical Connector

The Medical Connector is a standard connector used on some Freescale boards. It includes the most important analog peripherals and the channels for I2C communication. In the table below channels used for the glucometer implementation are shown.

Table 1. Channels used for glucometer implementation

1	VCC (3.3V)	VSS (GND)	2
3	—	—	4
5	ADC1	—	6
7	ADC2	Vref .1v DAC	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
19	TRIAMP 1 Out	TRIAMP 2 Out	20

3.1.2 Glucometer sensor (test strips)

The sensor used has an electroenzymatic approach, which means that it takes advantage of glucose oxidation with a glucose oxidase enzyme. The presence of glucose oxidase catalyzes the chemical reaction of glucose with oxygen, which causes an increase in pH, decrease in the partial pressure of oxygen, and increase of hydrogen peroxide because of the oxidation of glucose to gluconic acid:

$$\text{Glucos}e + O_2 \xrightarrow{\text{GlucoseOxidase}} \text{Gluconicacid} + H_2O_2$$

The test strip measures changes in one or several of this components to determine the concentration of glucose. The strips used in this design have three terminals or electrodes. Figure 2 shows the test strip terminals:

- Reference electrode
- Working electrode
- Trigger electrode

A negative voltage of –0.4 V is applied at the reference electrode. When blood or a glucose solution is placed in the strip, a chemical reaction occurs inside it, generating a small electrical current proportional to the glucose concentration. This current is constantly monitored while the strip is in place, allowing the device to monitor when blood is placed.

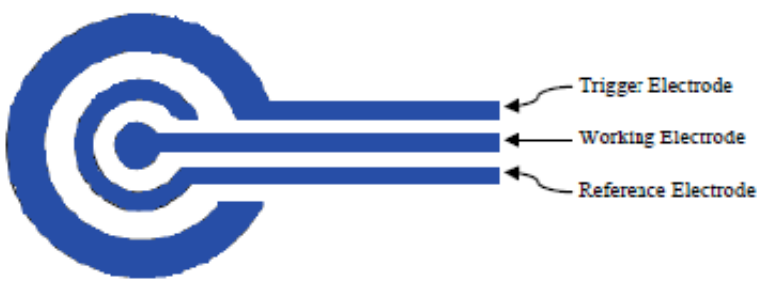

Figure 2. Test strip terminals

After the chemical reaction stabilizes, 5 s, the voltage is read by the ADC and compared using a look-up table to obtain the proportional glucose value in mg/dL. This value is sent to the host computer to inform the glucose value.

3.1.3 Current to voltage converter

The output generated by the test strip is a current that represents the glucose concentration. This current must be converted to voltage so that it can be properly filtered and treated. This conversion is performed using a current to voltage converter that has a single supply, low-input offset voltage, low-input offset and bias current TRIAMP embedded on the K53, and an external feedback resistance. The used circuit is shown in the figure below.

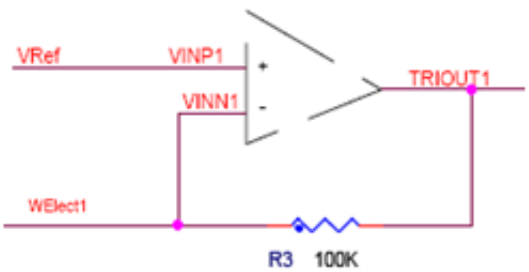

Figure 3. Current to voltage converter

The formula for the output voltage is:

$Vout = WElectlCurrent * R3$

3.1.4 Amplification and filtered

This block is divided into an amplification section and then a low pass filter with a cutoff frequency of 8 Hz designed to eliminate high frequency noise. The figure below shows the circuit.

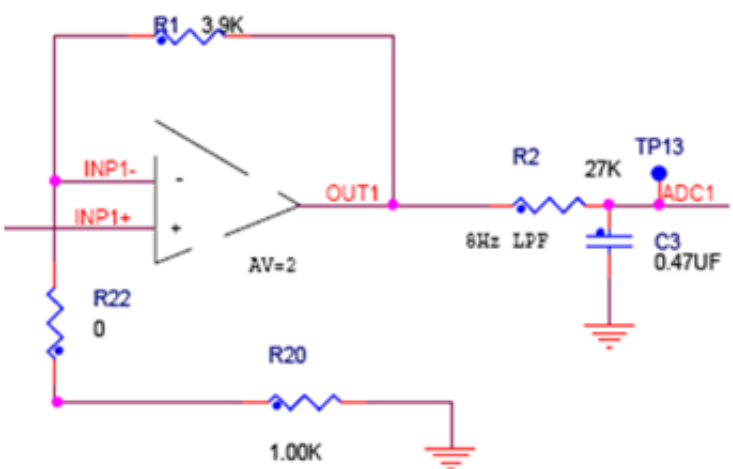

Figure 4. Amplification and filtered circuit

The following equation obtains the filter cut frequency using R2 and C3:

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

3.1.5 Vref generator 0.1 V

This reference voltage is generated by a simple voltage divisor and an external OPAMP. The figure below shows the Vref generator circuit (0.1 V).

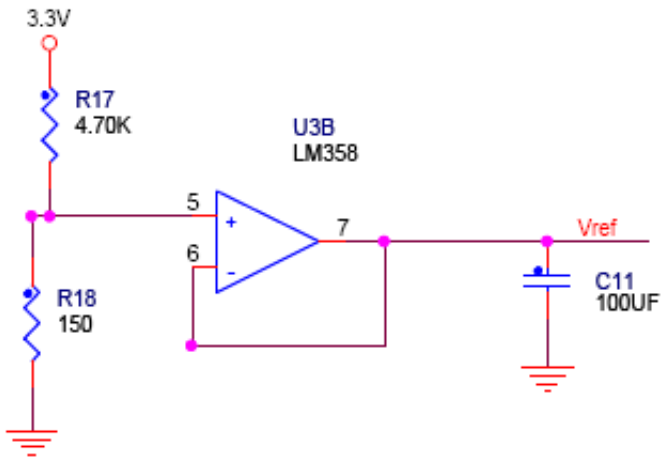

Figure 5. Vref generator circuit (0.1 V)

3.1.6 Vref generator –0.4 V

This negative voltage is applied, –0.4 V, in the reference electrode of the strips. First we obtain the negative voltage, –3.3 V, then with the help of a voltage inverter, a charge pump voltage inverter can be build with only three external 1µF capacitors.

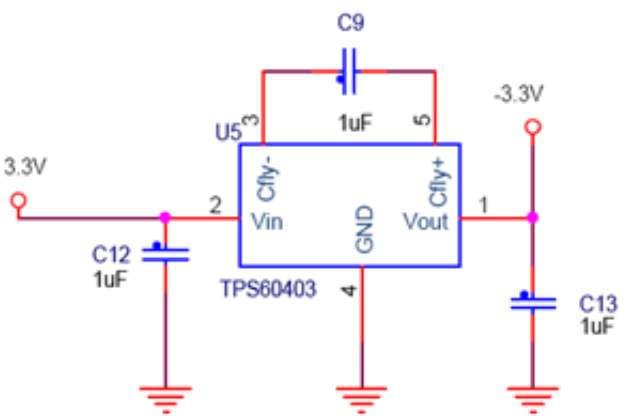

Figure 6. Voltage inverter circuit

With the negative voltage, the reference voltage, –0.4 V, is generated by a simple voltage divisor and an external OPAMP that is configured as a voltage follower. Following figure shows the Vref generator circuit, –0.4 V.

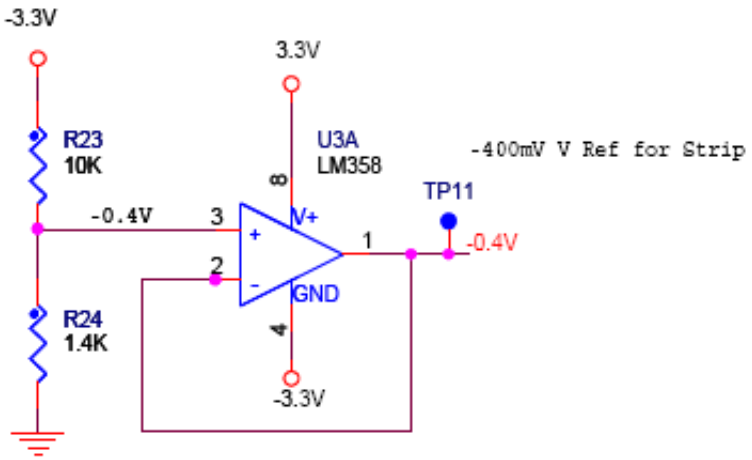

Figure 7. Vref generator circuit, –0.4 V

4 Software model

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

The figure below shows the general software model.

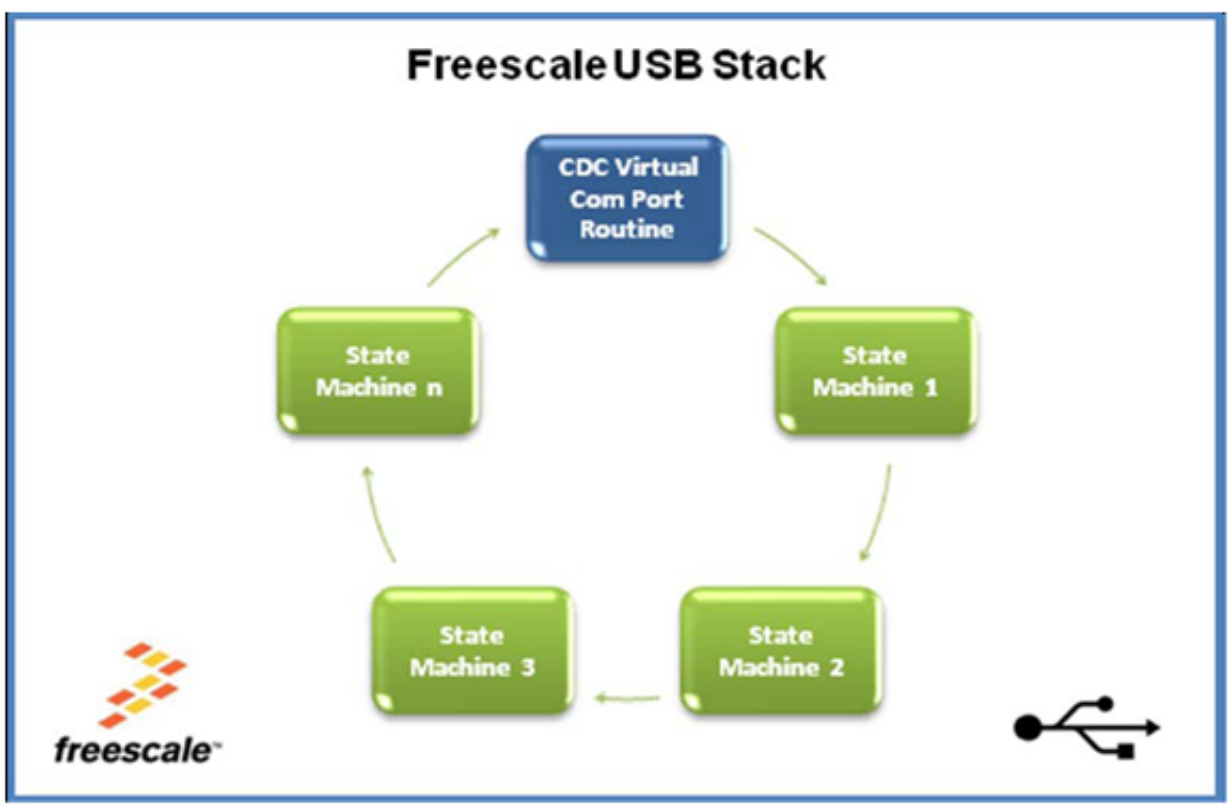

Figure 8. General software model

Each state machine is a task that has to be performed by the MCU. 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 substate 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 on **freescale.com/medicalusb.**

The MED-GLU software is divided into three main parts:

- Initialization
- Communication with PC
- Measurement execution

4.1 Initialization

When running the MED-GLU demo, the first step is to initialize all the peripherals needed for demo execution. On the main() function, the first function that is called Init_Sys(), initializes the clock and interrupts for working with USB. Then, some peripherals for AFEs and the Software Timer are initialized for the first use. USB is initialized as a CDC so communications with the host can start. After this, the state machines execute in an infinite loop.

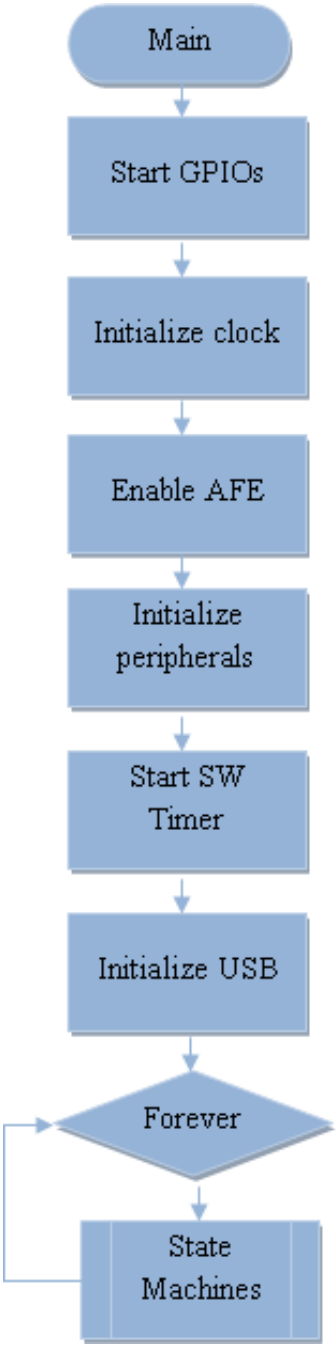

Figure 9. Initialization routine

4.2 Communication with PC

The communication of the device with the PC is via USB. The device is configured for working as a CDC and behaves as a Virtual Com Port installed on the computer.

4.2.1 Command reception

The main function for command reception is SerialComm_PeriodicTask, which is the CDC Virtual Com Port routine and is called by the main program. This function is constantly checks the USB input buffer for received data. When a data packet is received, the function checks whether the received packet is a request according to the communication protocol. If it is a request, the function checks the command requested and executes it. The figure below shows the flow diagram of the function SerialComm_PeriodicTask.

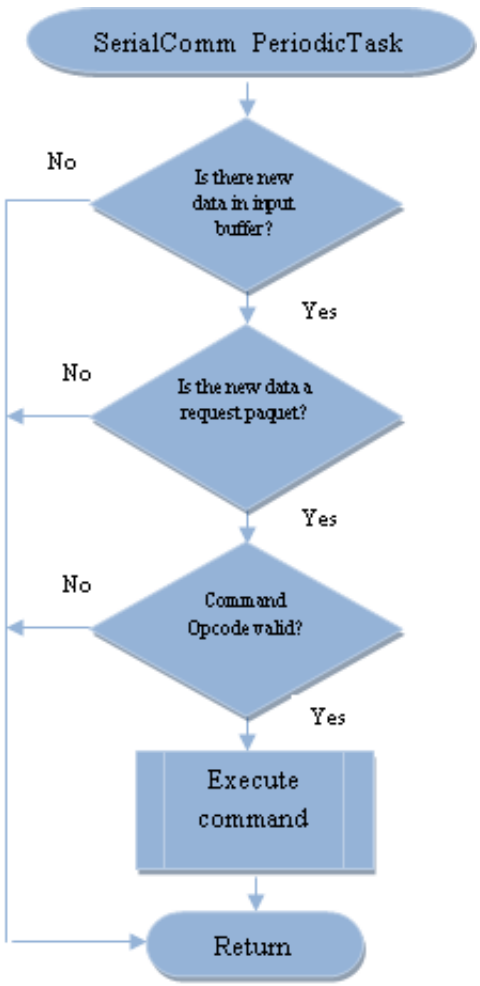

Figure 10. SerialComm_PeriodicTask flow diagram

4.2.2 Command execution

The MED-GLU demo software recognizes two Request Commands, GluStartMeasurementReq and GluAbortMeasurementReq, which start or stop glucometer measurements respectively. At the execution of either of these commands according to the Communication Protocol, a confirmation packet is generated indicating that the command has been received. When the GluStartMeasurementReq is executed, a confirmation packet is also generated to indicate whether the command was successfully executed or not. The figure below shows a request command flow.

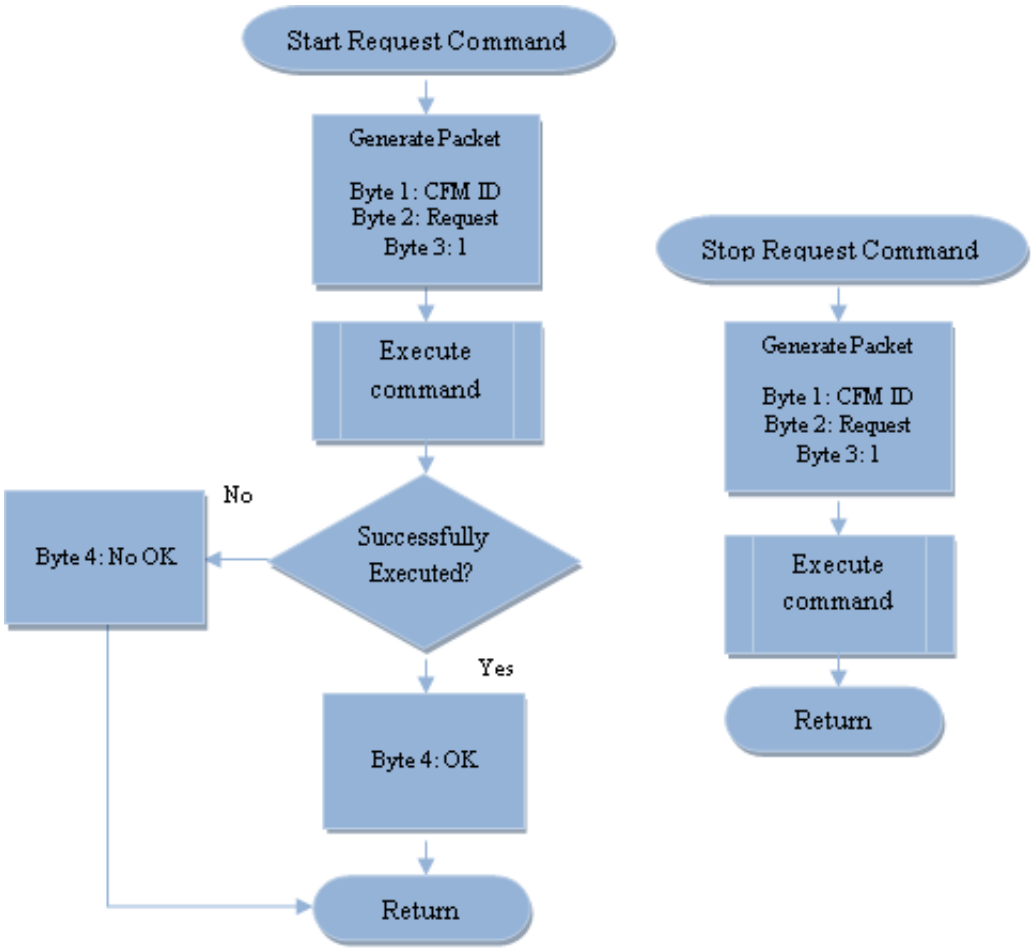

Figure 11. Request command flow

4.2.3 Sending packets

The function in charge of sending the data packets to the host is SerialComm_SendData. Data packets when created are stored on the output buffer and a data counter variable increases indicating the size of the output buffer. The function when called, checks the size of the data counter variable and if it is not zero, it means that there are new data packets in the output buffer that need to be sent. The function calls the CDC interface, part of the USB Stack with PHDC to send the packet. The figure below shows the flow diagram of the function SerialComm_SendData.

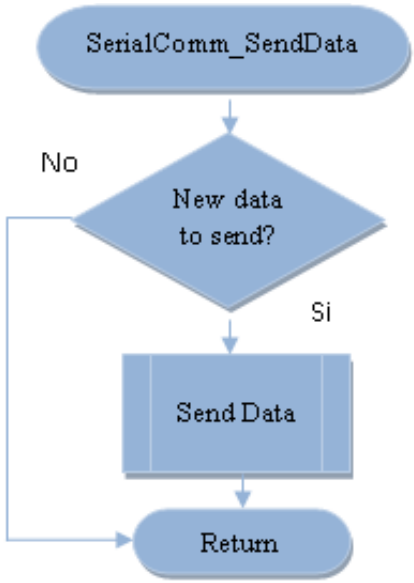

Figure 12. SerialComm_SendData flow diagram

4.3 Measurement execution

At the execution of the function GluStartMeasurementReq, the function GlucoseMeter_StartMeasurement is called. GlucoseMeter_StartMeasurement initializes what is needed, so measurements can be performed. This function first initializes ADCs for working with an 8-bit resolution, then resets the GlucoseMeter_GlucoseValue variable.

Then the GlucoseMeterActualState is set to STATE_WAITING_FOR_BLOOD, indicating to the GLU state machine that the glucose sample needs to be placed on the test strip. A Software Timer is now started for taking an ADC sample every 50 ms. More information about the Software Timer can be found in Software timer.

4.3.1 State waiting for blood

In the function GlucoseMeter_StartMeasurement, the GlucoseMeterActualState is taken out of its idle state and set to State Waiting for Blood. On the next state machine execution, the function StateWaitingForBlood is called.

The StateWaitingForBlood function asks whether a glucose solution has been detected by checking if the variable Electrode1 is more than a specific threshold, and an infinite loop is created until a glucose solution is detected. Once the detection has been made, the GlucoseMeterActualState is set to STATE_MEASURING, the GlucoseMeterActualEvent to EVENT_GLU_BLOOD_DETECTED and a timer of 5 s is started.

4.3.2 State measuring

When the 5 second timer is started in the StateWaitingForBlood function, the FinishGlucoseMeasurement_TimerEvent function is called, because that is the time the test strips require to stabilize the chemical reaction. After 5 seconds have elapsed, ADC values of the Electrode1 are used in a look-up table to obtain the proportional glucose measurements in mg/dL. This measurement is sent to the GUI to be displayed.

4.4 Calibration

The look-up table referred to is the result of a calibration process that has to be done before correct measurements are achieved. It has to be done with normal and constant conditions of temperature and humidity.

What is needed to do this calibration, is a simple commercial glucometer, glucose solution, and MED-GLU demo running. The following paragraphs explain in details about this calibration.

With the glucose solution, make four different concentrations in mg/dL from lower ones like 30 or 40 approximately, medium ones like 80 or 90, above medium like 130 or 140, and higher ones like 180 or 190. They do not need to be those values exactly, but it could be done with the help of the commercial glucometer and dilution techniques. After that, with the help of a serial terminal and the understanding of the Communication Protocol in Communication protocol, fill out a table as shown below; the values displayed are only examples.

Table 2. Calibration table example

Glucose value (mg/dL) Commercial glucometer (X)	ADC decimal value MED-GLU demo (Y)
33	55
99	126
130	150
157	175

The following figure shows a chart of the data made with Microsoft Excel.

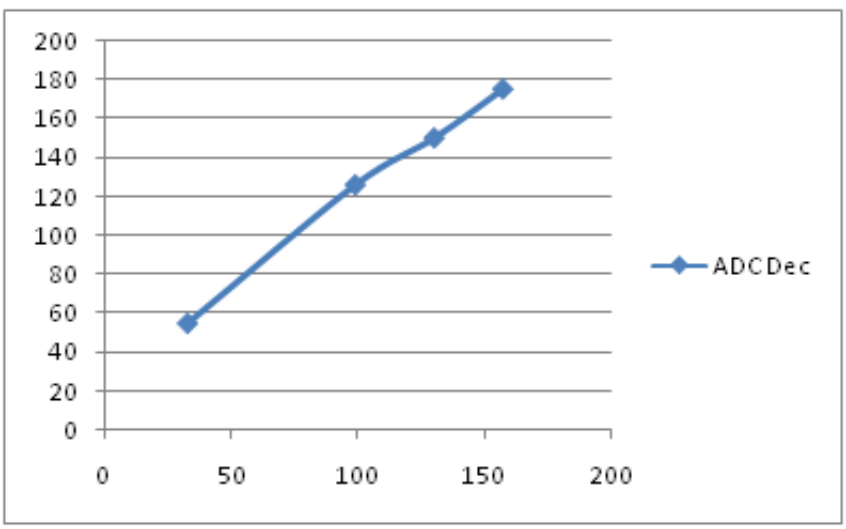

Figure 13. Correlation chart (commercial glucometer vs MED-GLU demo)

The behavior of the test strips is linear, so the results have to be extrapolated in order to get the desired look-up table. Right-click the blue data series in the chart, from the pop-up menu, select "Add Trendline..." In the Trendline Options, there are six types of trendlines available, select “Linear” and at the bottom select “Display Equation on chart” then click "Close". Following figure shows the chart with the trendline.

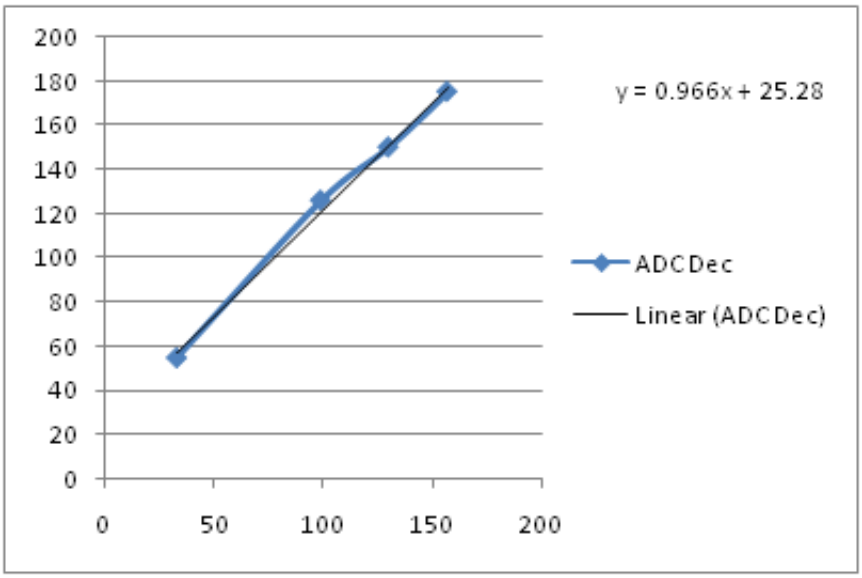

Figure 14. Chart with trendline

With equation y = 0.966x + 25.28, it is possible to extrapolate from an ADC value of 0 to 255, and in that way get a look-up table. The variable is GlucoseLookUpTable for Vref of 3 V.

5 Running MED-GLU demo

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

5.1 Tower system configuration

The following figure shows the components the MED-GLU demo that need to be assembled to work properly.

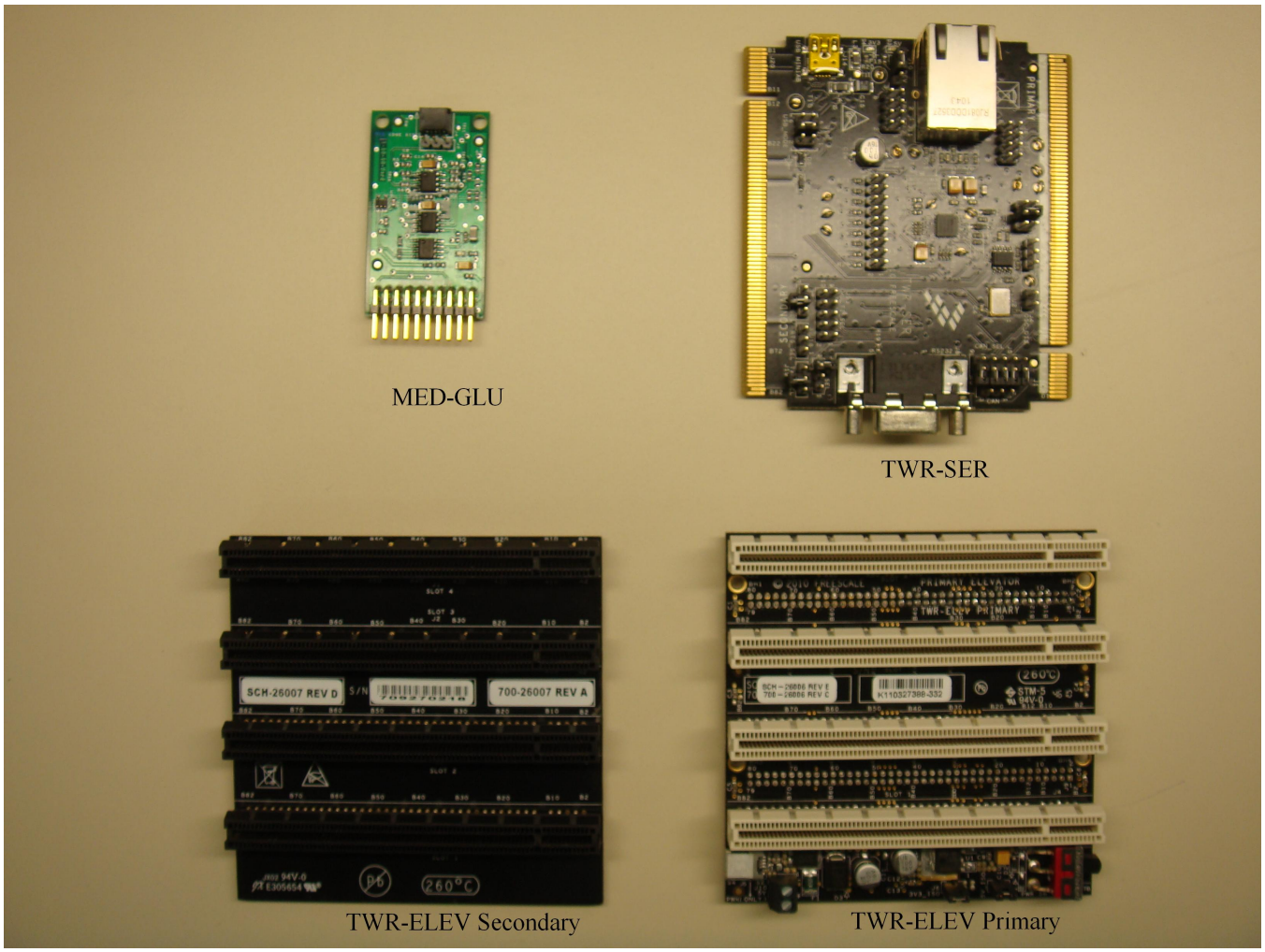

Figure 15. Required components

The TWR-SER, TWR-K53N512, TWR-MCF51MM, and TWR-S08MM128 boards require the configuration of some jumpers. The jumper configuration list is the following:

TWR-SER
- J10 1–2
- J16 3–4

TWR-K53N512
- J1 Disconnected
- J3 Disconnected
- J4 1–2
- J11 1–2
- J15 1–2
- J17 1–2
- J18 1–2

TWR-MCF51MM and TWR-S08MM128 jumper configuration:
- J1 1–2
- J2 1–2
- J10 2–3
- J11 1–2

- J12 Open
- J18 1–2, 9–10, 11–12, and 13–14 Open

When the jumpers have been configured, the Tower System has to be assembled, the connector marked as PRIMARY with the primary elevator, and the connector marked as SECONDARY with the secondary elevator. The MED-GLU AFE must be connected on the medical connector. The following figure shows the Tower System assembled.

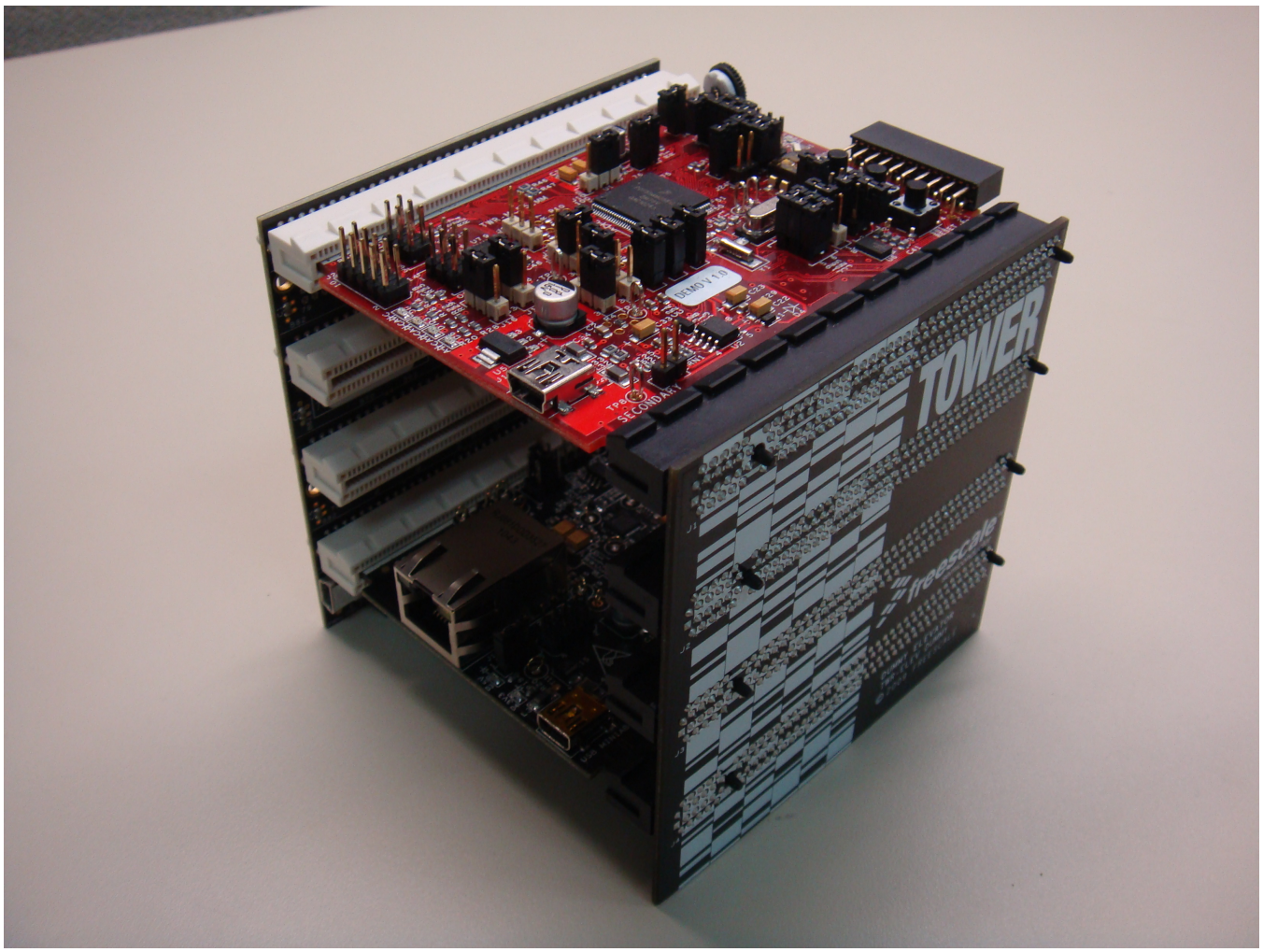

Figure 16. Tower system

5.2 Loading application

Software for the MED-GLU demo is developed on IAR Workbench for ARM 6.10 for the Kinetis K53 MCU and on Freescale CodeWarrior 6.3, Service Pack for compatibility with MM/JE devices required, for MCF51MM256 and MC9S08MM128 MCUs. This section explains how to load the program on each IDE.

5.2.1 Loading IAR project on K53

1. Connect the USB cable to the TWR-K53N512 board. Make sure the device is recognized as an Open Source BDM – Debug Port. Install the Open Source BDM drivers in the IAR folder.

2. Open the project, USB_CDC.eww, and make sure the debugger in the project options panel is PE micro. To do that, choose Menu > Project > Options, select the category Debugger and check that PE micro is selected as shown in figure below.

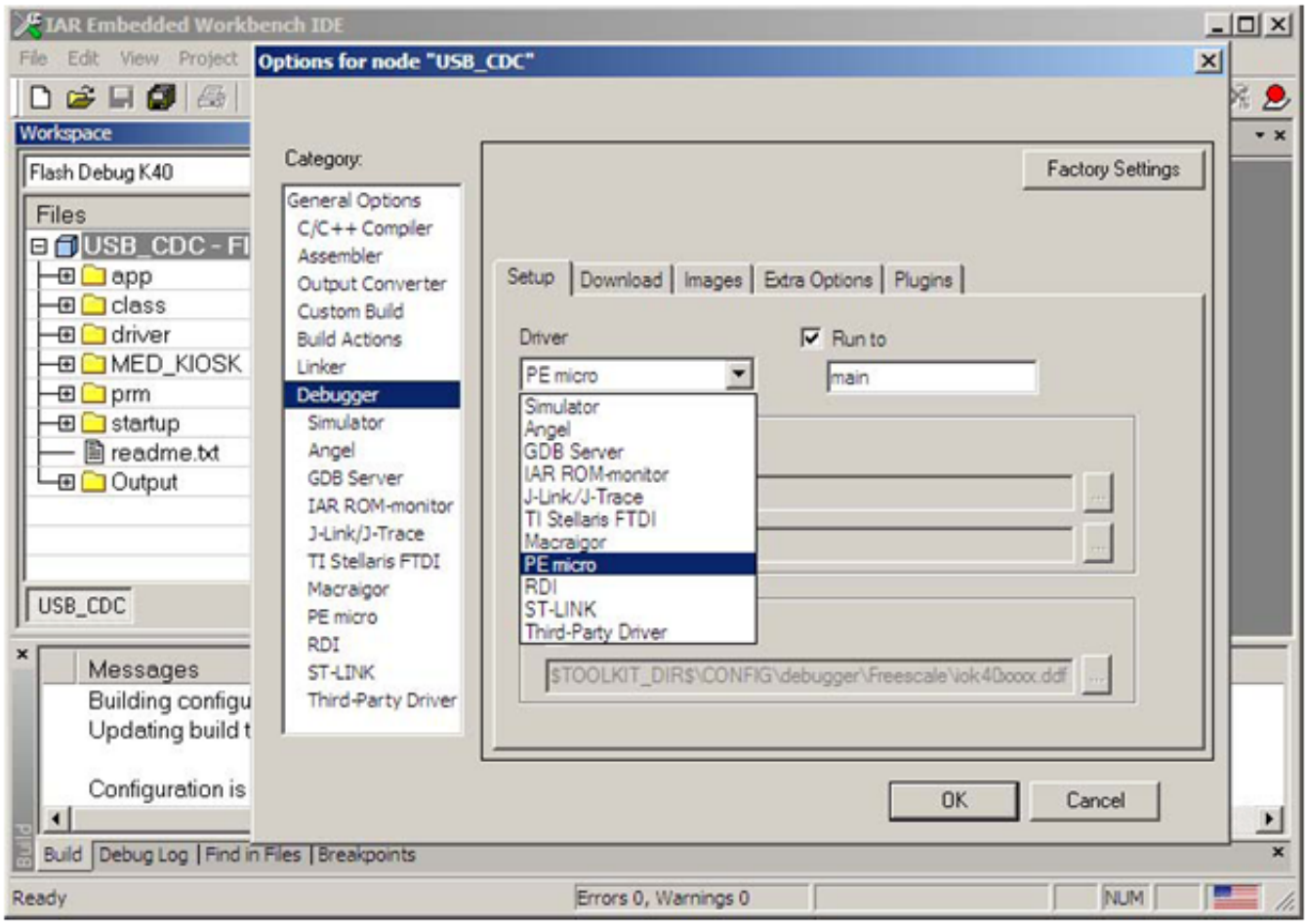

Figure 17. Debugger selection

3. By clicking the DEBUG button, the project will be loaded into the MCU (see figure below). The project will compile automatically and load into the MCU. After that, disconnect the USB cable from the TWR-K53N512 board.

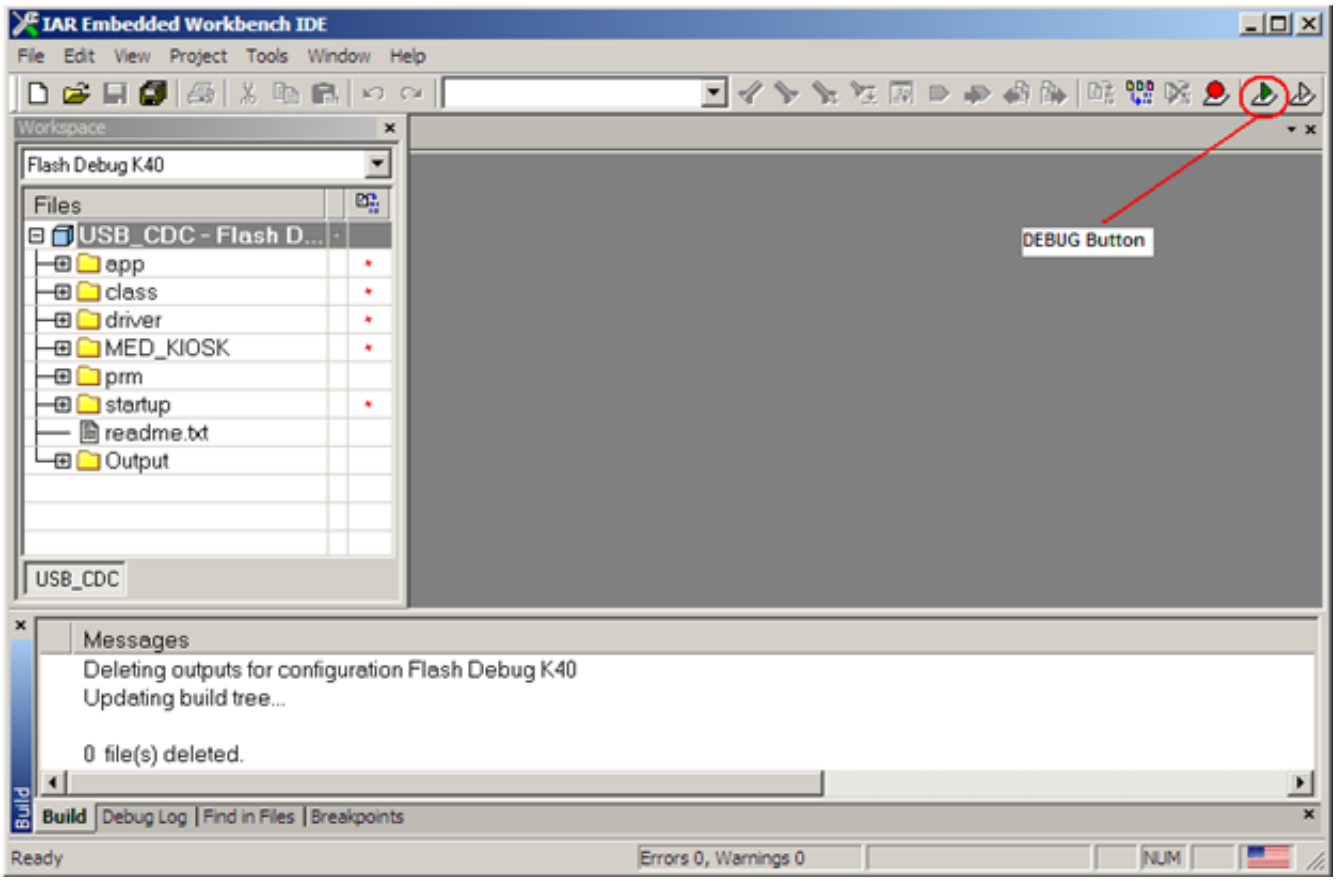

Figure 18. Debug button

5.2.2 Loading CodeWarrior project on MM devices

1. Connect the USB cable to the TWR-SER board to turn it on. 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 make sure the option Open Source BDM is selected for debugging as shown in Figure 19.
3. Press the Debug button to automatically compile and load the program into the MCU.

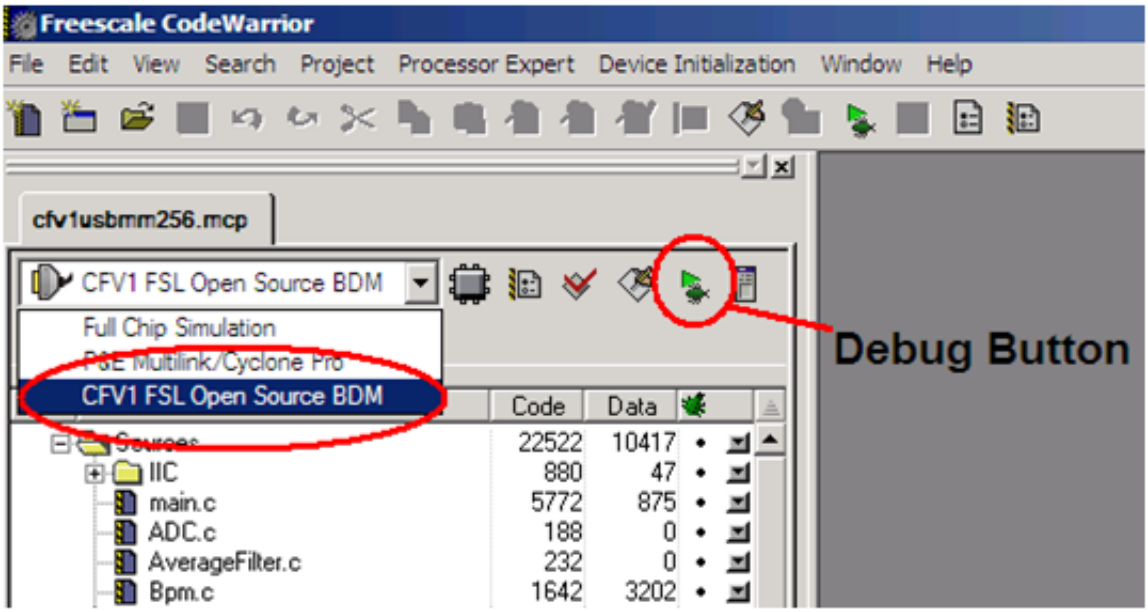

Figure 19. CodeWarrior window

4. When the program has loaded successfully into the MCU, disconnect the USB cable from both the TWR-SER and MM controller board.

5.3 Running MED-GLU demo

After the Tower System is assembled and the project loaded in the MM or K53 MCU, the demo can be used. Follow the next steps to run the demo without problems:

1. Connect the USB cable to the TWR-SER board as shown in Figure 20. The device has to be recognized as a Virtual Com Port. To accomplish this, install the driver for the Virtual Com Port included in the project folder.

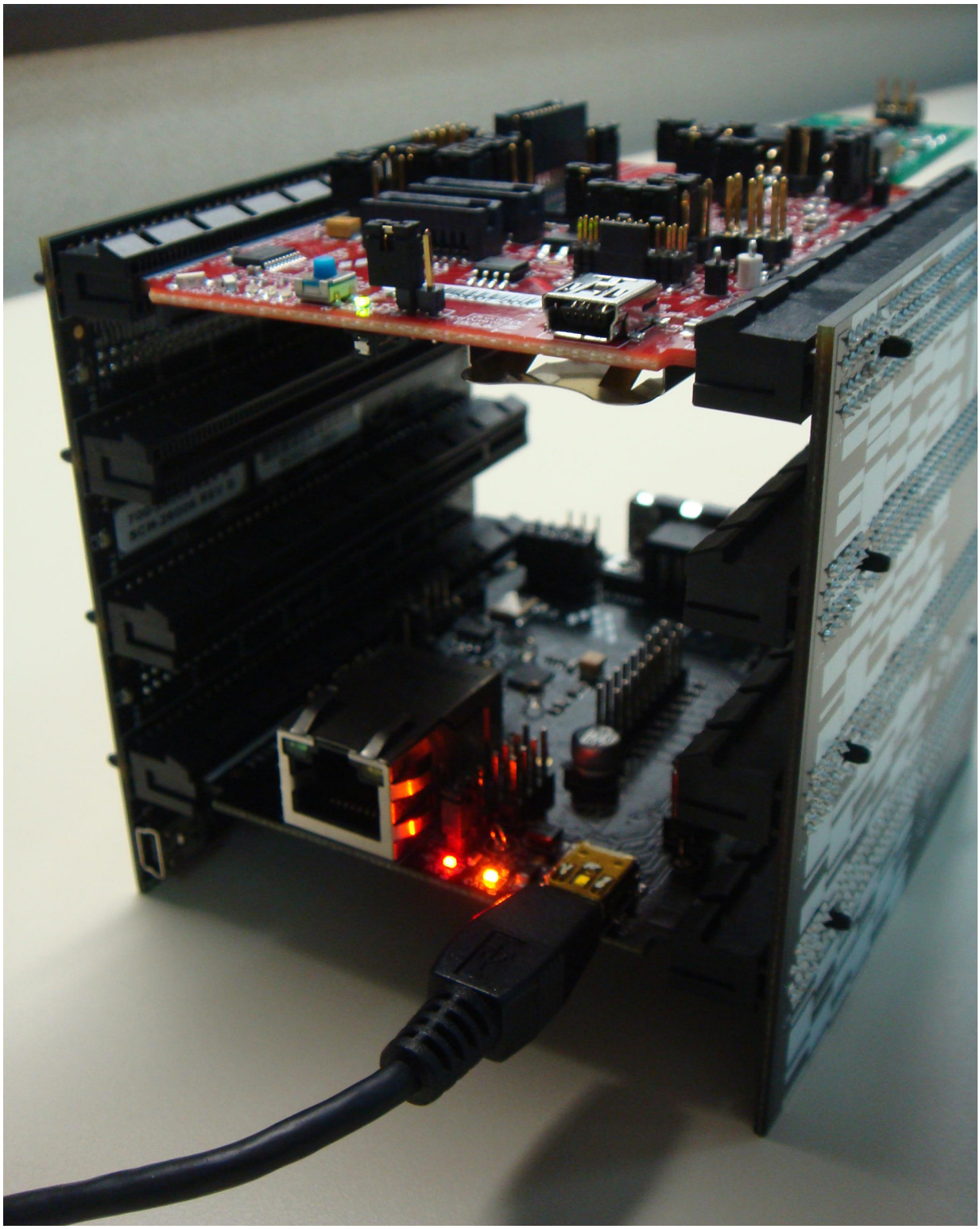

Figure 20. USB cable connected to TWR-SER

2. After the driver has been installed and the device recognized correctly, open the Graphic User Interface (GUI). A little window called “Port Chooser” will appear asking for the COM PORT number assigned to the device.
3. Select the correct COM PORT number and click OK as shown in Figure 21, the GUI interface will appear on the screen as shown in Figure 22.

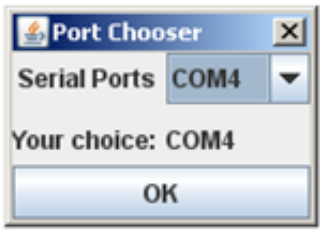

Figure 21. Port Chooser

Figure 22. GUI Main screen

4. Make sure the Caps Lock is not activated on the keyboard, and then press Shift + D to enter the GUI in doctor mode as shown in figure below.

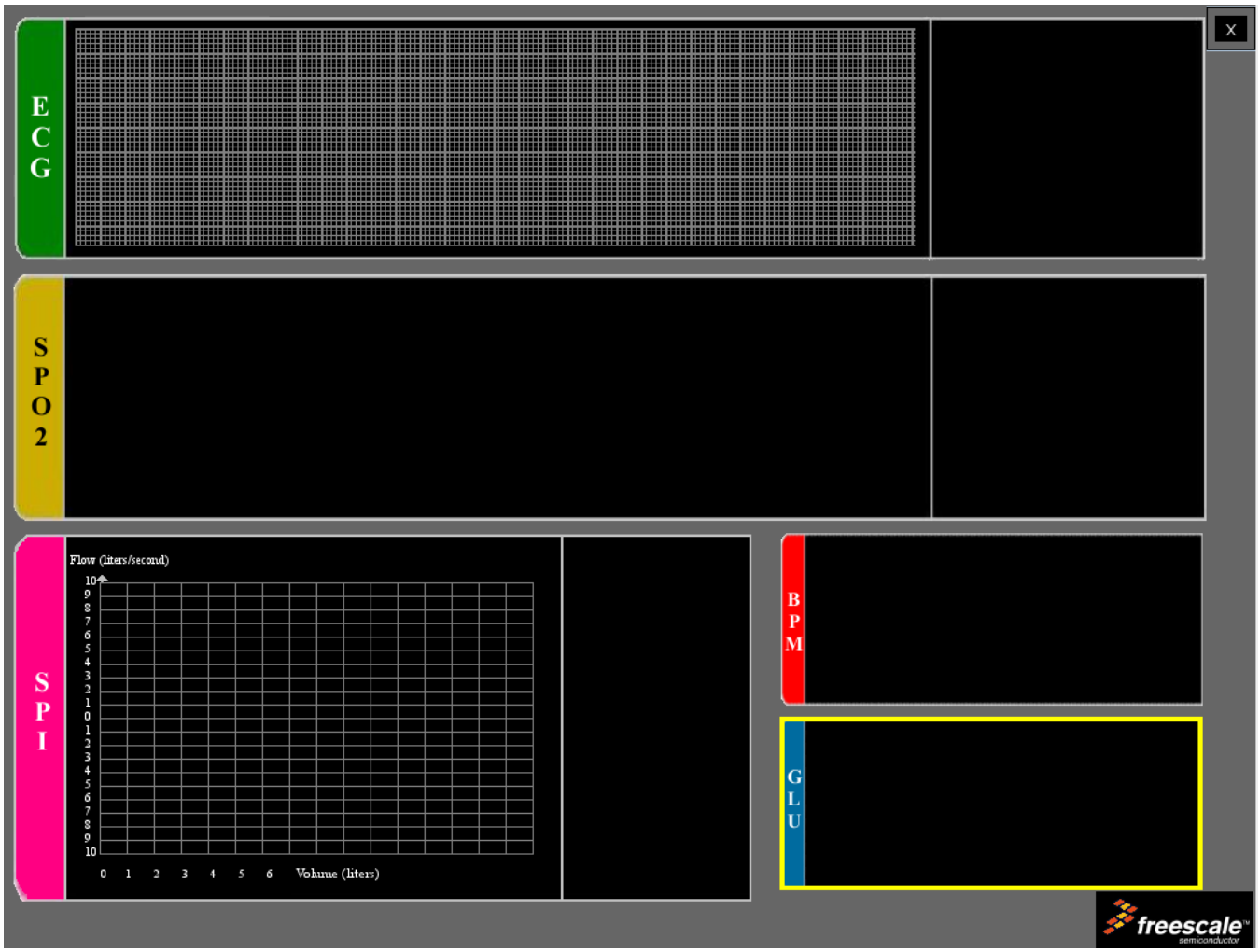

Figure 23. GUI in doctor mode

5. Place the test strip on the connector of the AFE as shown in Figure 24 and click on the GLU area on the GUI. Now place the glucose sample on the white target area at the end of the test strip as shown in Figure 25. Measurements will start and after a while, the measurements will appear on screen as shown in Figure 26.

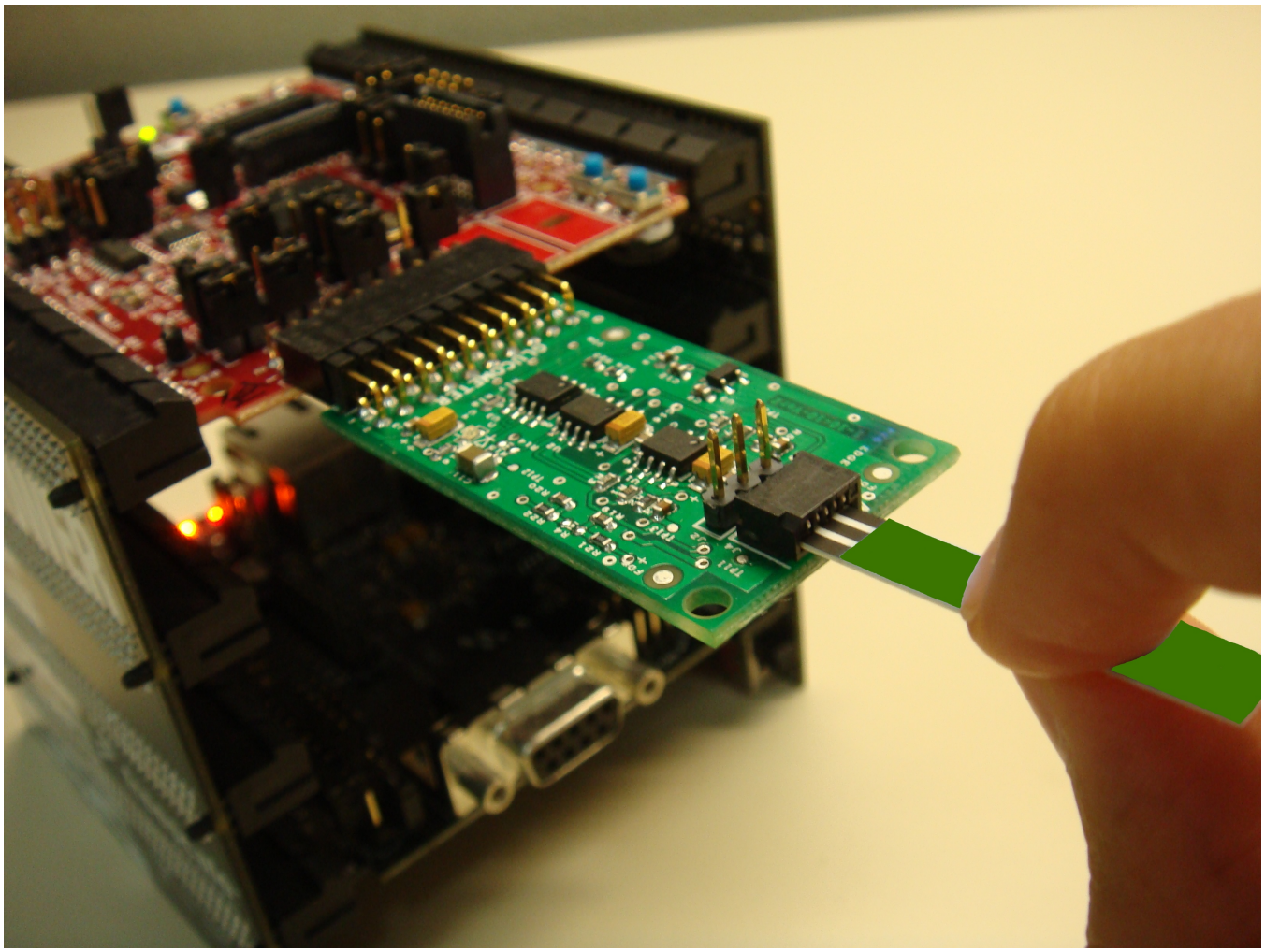

Figure 24. Test strip placement

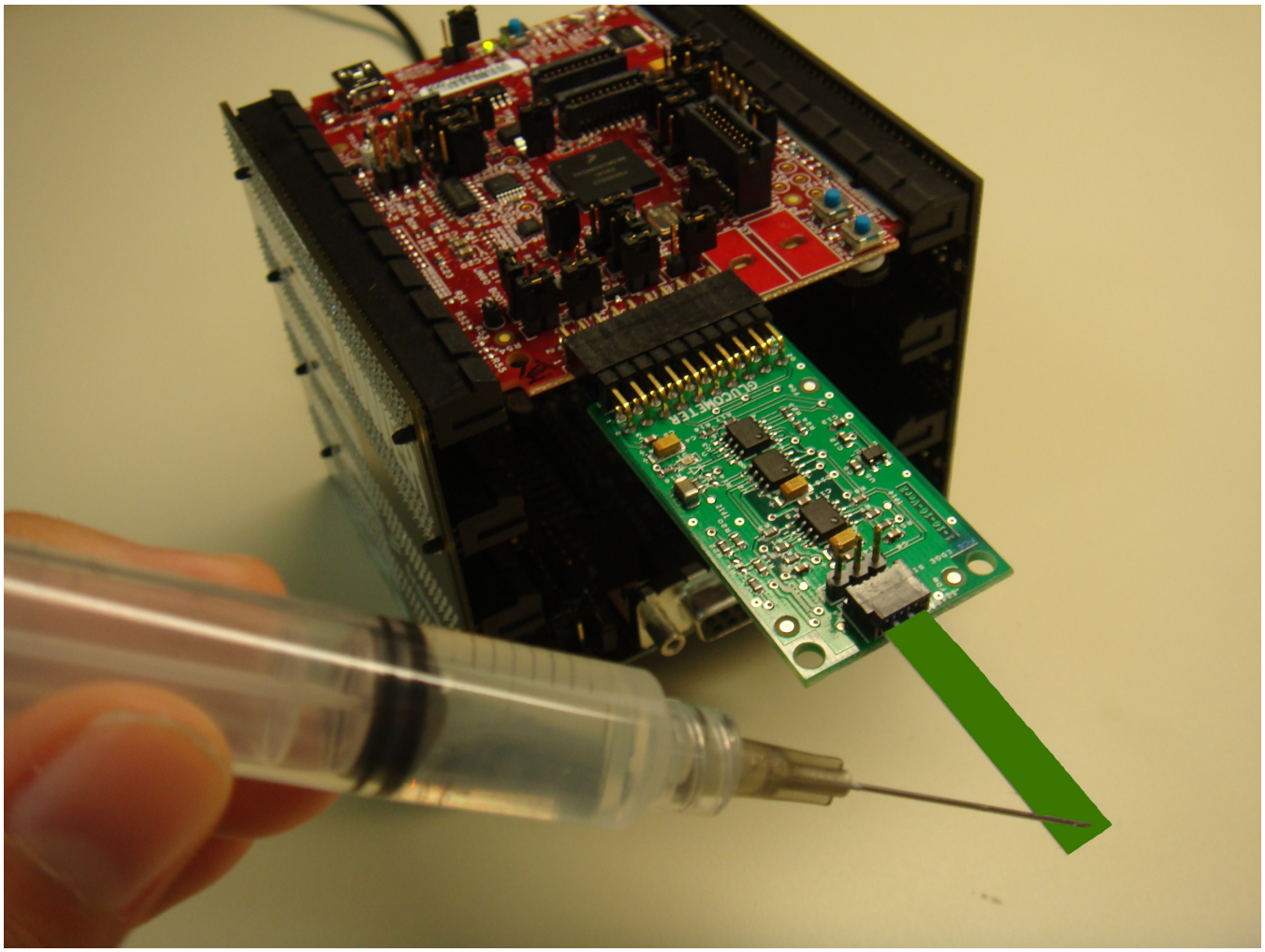

Figure 25. Glucose sample placement

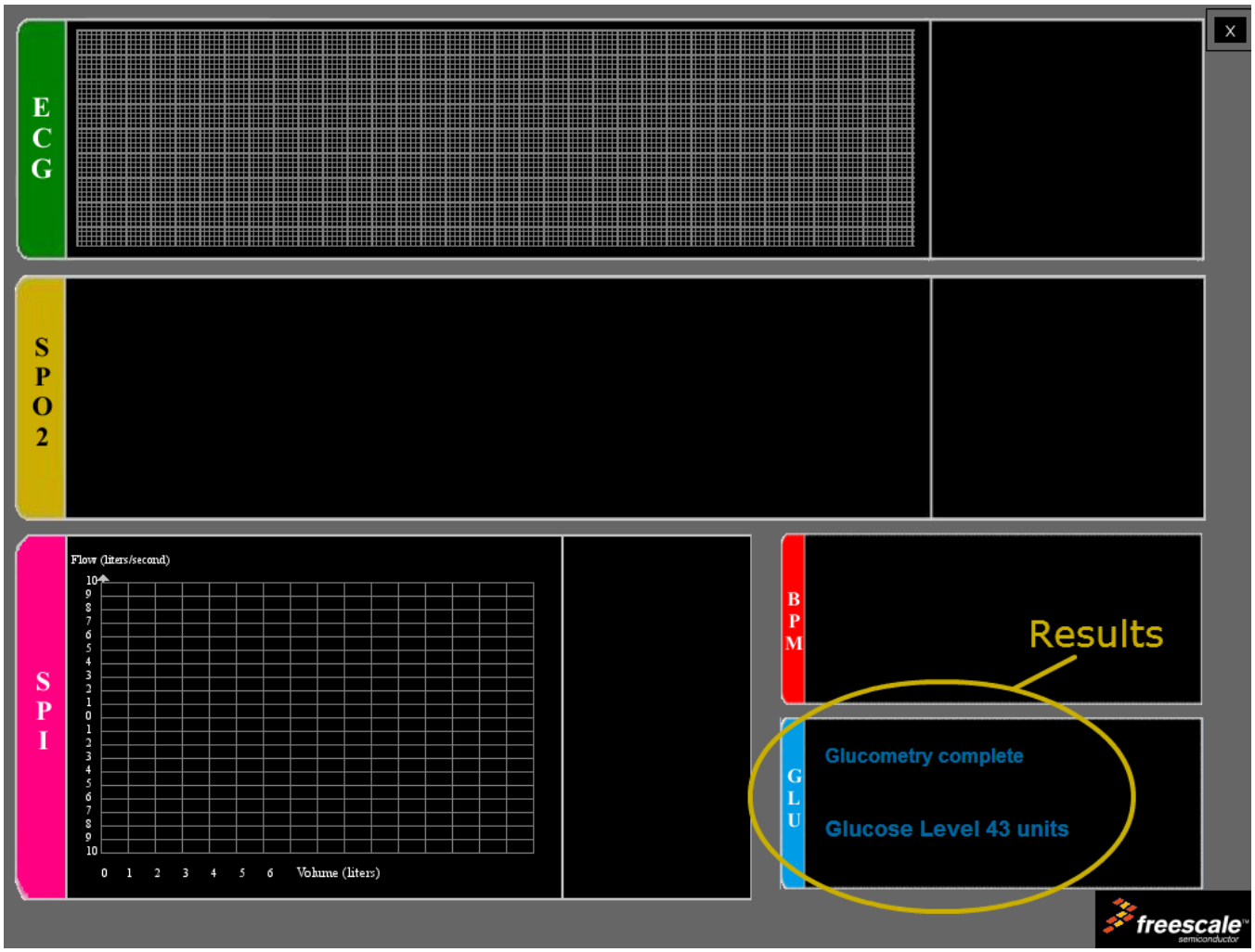

Figure 26. GUI running

6 References

This section lists online resources related to this application note, available on the Freescale website.

- Reference manuals and data sheets for Kinetis K53 (*K53P144M100SF2RM*, *K53P144M100SF2*), MCF51MM256 (*MCF51MM256RM*, *MCF51MM256*), and MC9S08MM128 (*MC9S08MM128RM*, *MC9S08MM128*).
- Schematics of the Tower Modules and MED-GLU board.
- USB Stack PHDC User Guide (*MEDUSBUG*).

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.

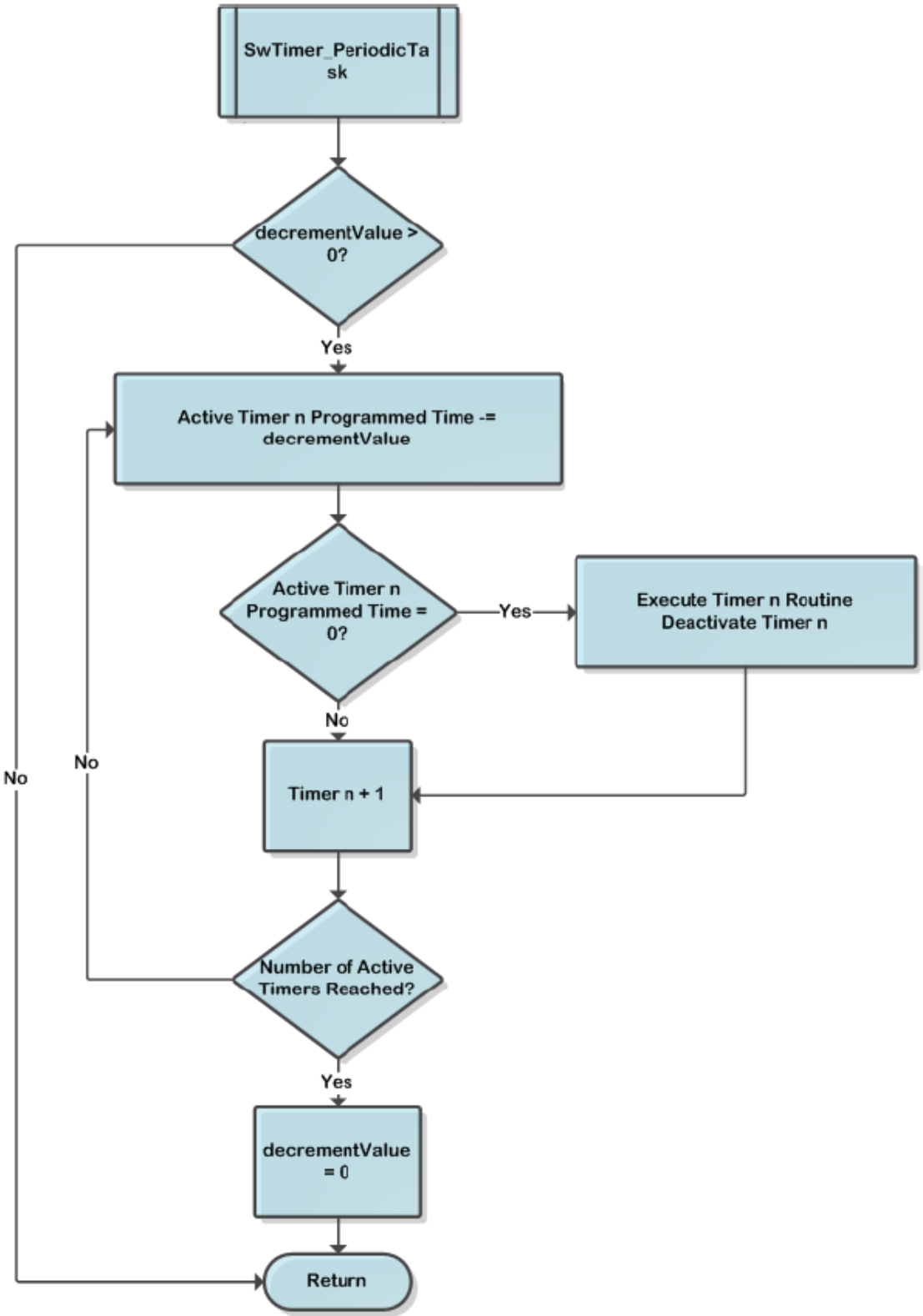

Figure A-1. SwTimer_PeriodicTask subroutine

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.

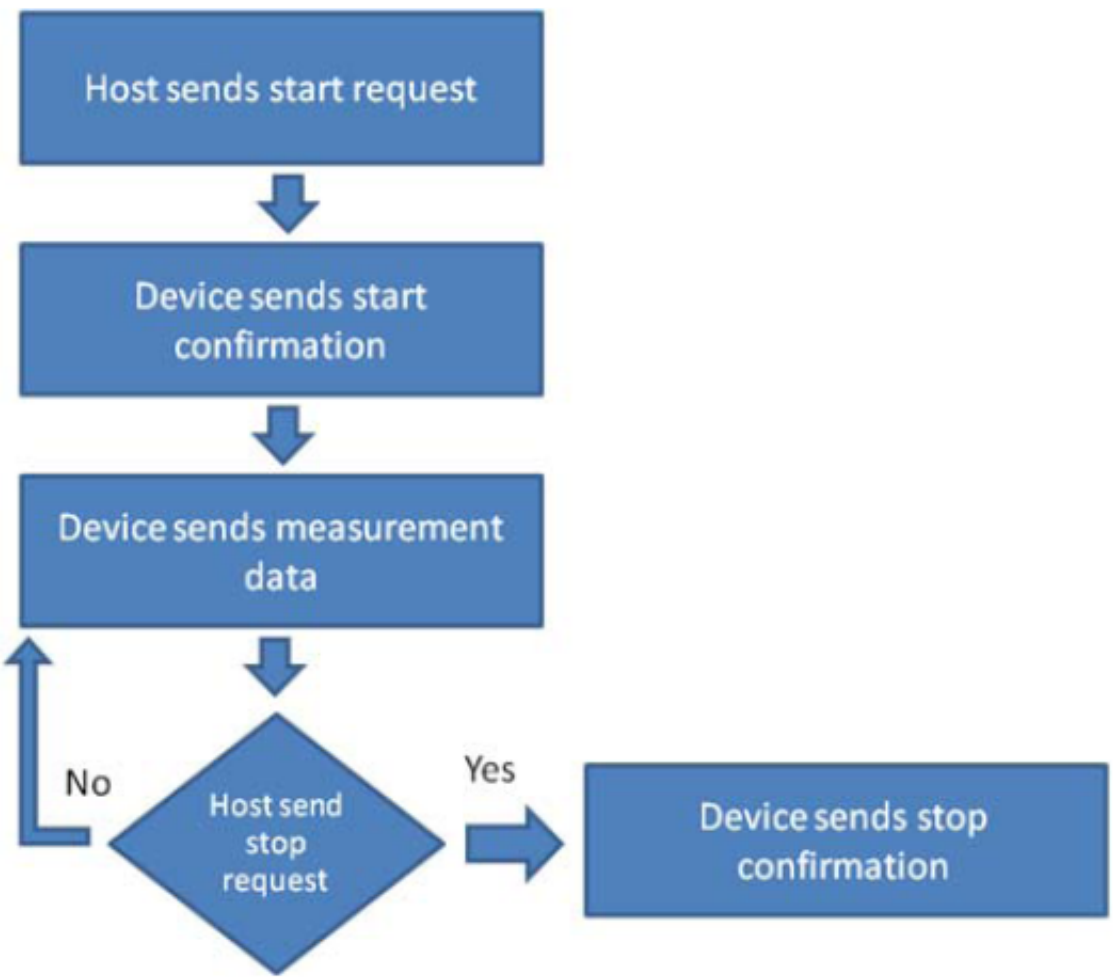

Figure B-1. Communication protocol data flow

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

- Packet Type
- Command Opcode

- 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[1]

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
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

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_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
SPO2_DIAGNOSTIC_MODE_NEW_DATA_READY			X	0x27
				0x21
System commands				
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.

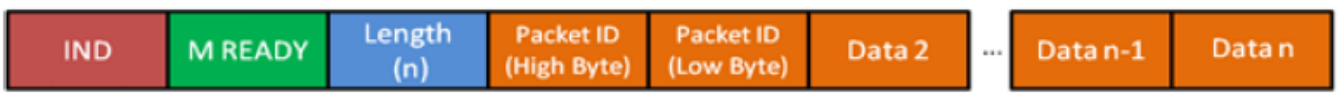

Figure B-6. Indication packet structure

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.

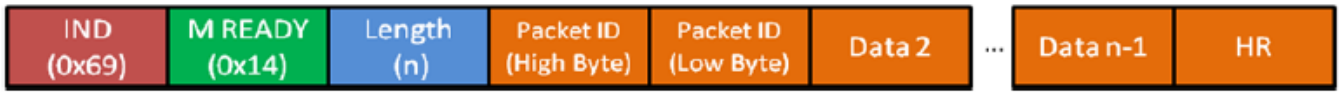

Figure B-7. MED-EKG IND packet

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

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: AN4364
Rev. 1, 01/2013