Freescale Semiconductor
Application Note

Document Number: AN4496
Rev. 0, 03/2012

Pulse Oximeter Using USB PHDC

by: Jose Santiago Lopez Ramirez
RTAC Américas

1 Introduction

This Application note explains the implementation of a pulse oximeter which communicates with a computer using the USB Personal Healthcare Device Class. Implementation is done on the Freescale MK53N512 Kinetis microcontroller, but can be implemented on any Freescale USB capable microcontroller.

This Application note is intended for medical solutions developers, biomedical engineers or any person interested on the USB personal healthcare device class. Nevertheless, some skills in C programming and microcontrollers handling are required.

This Application note is closely related with the application note “AN4327 Pulse Oximeter Fundamentals and Design”. It is recommended to read AN4327 for better understanding.

2 Personal Healthcare Device Class Overview

Universal Serial Bus (USB) is a standard which defines hardware and protocols for intercommunication between a host (usually a PC) and one or more devices. Each USB device has its own purpose, and therefore, they are divided in different classes according to their function. One example is the Human Interface Device (HID) class which is used in devices like computer keyboards and mouse.

Contents

Personal Healthcare Device Class (PHDC) defines the requirements to establish communication and seamless interoperability between personal USB medical devices and USB hosts, to be then processed, stored or transmitted to a doctor or relative via internet.

USB PHDC is used by healthcare exchange protocols like ISO/IEEE 11073-20601 as the transportation method for the communication packets between the host and the personal healthcare device. It standardizes the way in which the data and messages are sent over USB.

3 Pulse Oximeter Implementation

Pulse Oximeter is implemented using Freescale TWR-K53N512, a tower development board including the medical oriented microcontroller MK53N512, MED-SPO2 an analog front end board for pulse oximetry solutions development and TWR-SER a tower system board for designs including serial communications. This is the same hardware used in AN4327 “Pulse Oximeter Fundamentals and Design”. Please refer to this application note for more information about pulse oximetry principles and the hardware used in the pulse oximeter development.

System is based on the Freescale USB stack with PHDC which is free code for developing solutions that require USB connectivity and can be downloaded from the Freescale web page. This stack contains functions that can be used at device level (configure clocks, initialize USB module, etc…) and the class level (send-receive packets, send descriptors, etc…). Please refer to Freescale USB Stack with PHDC Stack Users Guide and Freescale USB Stack with PHDC Device API Reference Manual for better understanding.

Software is basically divided in three main parts: System Initialization, Application Initialization and Application Execution. Final application is executed in an infinite loop as shown in the following flow diagram (Figure 1).

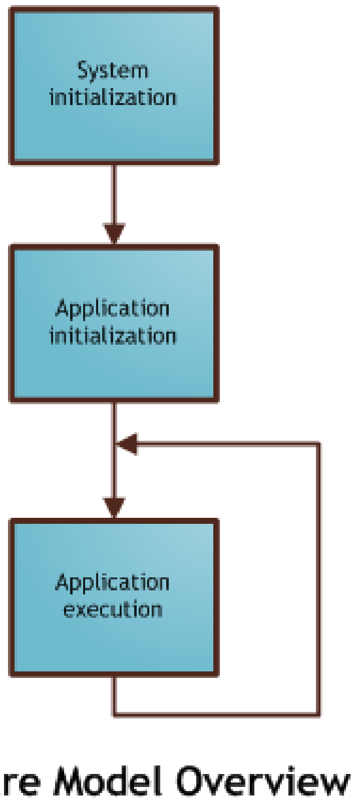


Figure 1. Software model flow diagram

For a better understanding of this chapter, it is highly recommended to open the MED-SPO2 PDHC C project and view it as you read these lines.

3.1 System Initialization

System initialization is executed when the function Init_Sys is called at the beginning of the program. Init_Sys is a device level function and varies upon the microcontroller. It initializes the required peripherals on the microcontroller for the stack functionality. Init_Sys first enables the interrupts on the USB module configuring the NVICICER2 and NVICISER2 registers. Then it enables the GPIO modules required by the microcontroller calling the function GPIO_Init. Init_Sys now calls pll_init function which configures the microcontroller for working at 50MHz using an external clock source. Once the microcontroller's clock has been configured, MPU_CESR register is cleared and microcontroller is configured to energize and bring clock signal to USB module for future enumeration.

3.2 Application Initialization

Application initialization configures the previously initialized modules for use of the pulse oximetry PHDC application. This configuration starts when the function TestApp_Init is called. TestApp_Init first calls the function PHD_Transport_Init. This function manages the enumeration of the microcontroller's USB module as PHDC by enabling the Pull-Up resistors and handling the enumeration process. PHD_Transport_Init returns an error value. If error "OK" is returned it means that the device has been already enumerated as a PHD (Personal Healthcare Device) otherwise something went wrong during the enumeration and device might not be recognized by the host PC. At this point, the device is recognized by the host as a PHD but it is not defined yet as a pulse oximeter device using the standard ISO/IEEE 11073-20601.

After enumeration, TWR-K53N512 on-board LEDs and push buttons are configured for future use. SwTimer_Init function is called for initializing the software timer. More information about the software timer can be found on the application note "AN4327 Pulse Oximeter Fundamentals and Design": Appendix A Software Timer.

The last function called is vfnSpO2_AFE_Init. This function initializes the required peripherals (OpAmps, TRIAMPs, ADCs and timers) required by the MED-SPO2 board.

3.3 Application Execution

Once the peripherals have been configured, a connection between the host PC and the device is established. Host PC recognizes the device as a PHD but it is not fully functional yet. A communication protocol between the host PC and the device is required in order to exchange the information in a standardized and reliable manner.

Several communication protocols exist, including some vendor specific protocols. Nevertheless, engineering bets on standardized protocols that ensure same interoperability between medical devices.

Continua Health Alliance® is an organization that promotes the enhanced interoperability among medical devices. The implementation of this demo is based on the Continua® standard for health data communication between host PC and device which uses the standard ISO/IEEE 11073-20601 "Personal health device communication: Optimized exchange protocol" as a base.

A brief explanation of the 11073-20601 communication protocol is shown below. For a complete explanation of the communication protocol refer to the ISO/IEEE 11073-20601 standard.

3.3.1 ISO/IEEE 11073-20601 communication process

The standard 11073-20601 defines the communication protocol between medical devices or "Agents" and hosts or "Managers".

The Agent can be defined as a set of objects called MDS (Medical Device System). Each MDS describes a behavior of the agent (e.g. pulse oximeter or blood pressure monitor). Each Agent can contain one or more of these MDS objects.

In the same manner, each MDS object contains sub-objects that define its behavior (e.g. measurements to report). All this information must be reported to the Manager so it can control the behavior of the Agent. Nevertheless only one MDS object must be reported at a time (e.g. an Agent cannot be a pulse oximeter and a blood pressure monitor at the same time). Following diagram represents an Agent capable of being a pulse oximeter and a blood pressure monitor).

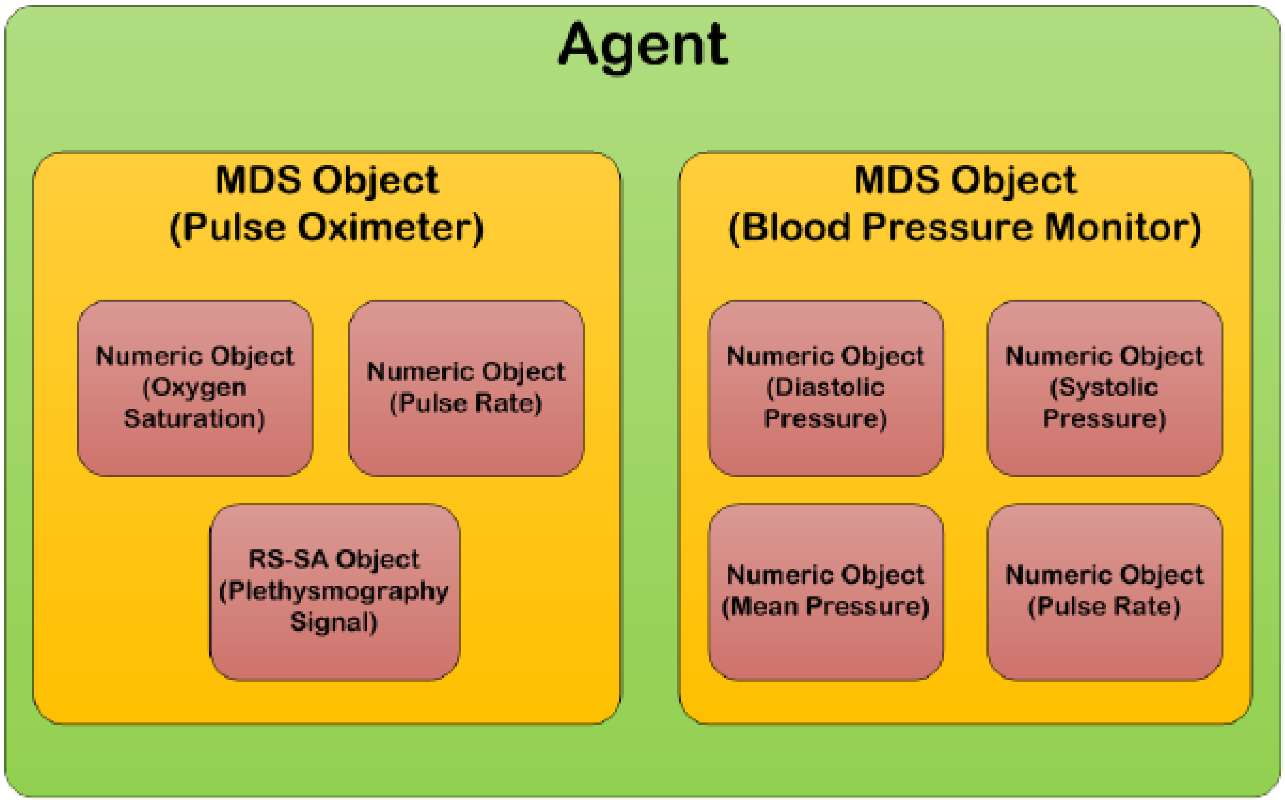

Agent
MDS Object (Pulse Oximeter)
Numeric Object (Oxygen Saturation)
Numeric Object (Pulse Rate)
RS-SA Object (Plethysmography Signal)
MDS Object (Blood Pressure Monitor)
Numeric Object (Diastolic Pressure)
Numeric Object (Systolic Pressure)
Numeric Object (Mean Pressure)
Numeric Object (Pulse Rate)

Figure 2. Agent representation

In the case of this demo, the Agent contains only one MDS object corresponding to the pulse oximeter application. More detailed information about the Agent representation can be found on the ISO/IEEE 11073-20601:2010 document, in the chapter 6, Personal health device DIM.

IEEE standard defines a state machine for the Agents and other state machine for the Managers. Since our demo application is a device, we will only explain the Agent’s state machine. Following diagram is a simplified representation of the state machine shown in the Chapter 8, Figure 10 of the ISO/IEEE 11073-20601:2010 standard.

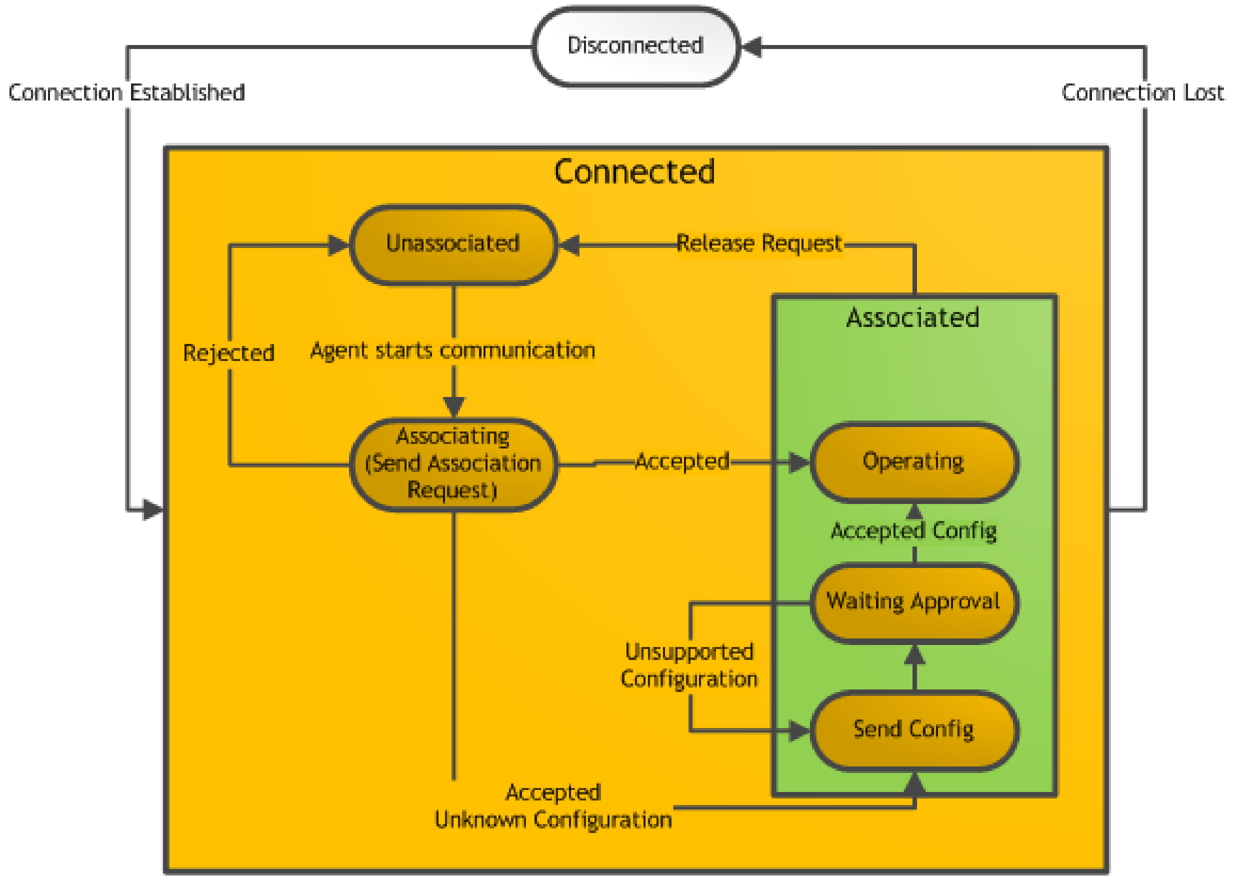


Figure 3. Agent's state machine

At the beginning, the Agent is disconnected from the Manager. Agent must be connected to the manager in order to establish a communication. When the connection has been established (in our case when the USB device has been enumerated as PHDC device) the Agent passes to be in a connected state.

Once connected, the Agent is initially in an "Unassociated" state. The Agent must send an "Association Request" to start communications. Association request is sent as an APDU (application protocol data unit), a data packet containing the required information to start the association and it must correspond to the MDS object to associate. The association request APDU must look like the following.

```
/* association request to send */
uint_8 USB_CONST PHD_OXI_ASSOC_REQ[ASSOC_REQ_SIZE] = {
0xE2, 0x00,                               /* APDU CHOICE Type (AarqApdu) */
0x00, 0x32,                               /* CHOICE.length = 50 */
0x80, 0x00, 0x00, 0x00,             /* assoc-version */
0x00, 0x01, 0x00, 0x2A,             /* data-proto-list.count=1 | length=42*/
0x50, 0x79,                               /* data-proto-id = 20601 */
0x00, 0x26,                               /* data-proto-info length = 38 */
0x80, 0x00, 0x00, 0x00,            /* protocolVersion */
0x80, 0x00,                              /* encoding rules = MDER or PER */
0x80, 0x00, 0x00, 0x00,            /* nomenclatureVersion */
0x00, 0x00, 0x00, 0x00,            /* functionalUnits |
                                               no test association capabilities */
0x00, 0x80, 0x00, 0x00,            /* systemType = sys-type-agent */
0x00, 0x08,                              /* system-id length = 8 and value
     ,                                          (manufacturer- and device- specific) */
0x4C, 0x4E, 0x49, 0x41, 0x47, 0x45, 0x4E, 0x54,
0x40, 0x00,                              /* dev-config-id | extended configuration*/
0x00, 0x01,                              /* data-req-mode-flags 0x00, 0x01*/
0x01, 0x00,                              /* data-req-init-agent-count,
                                                    data-req-init-manager-count */
0x00, 0x00, 0x00, 0x00             /* Atribute list */
};
```

When the Agent sends the association request, it goes to the "Associating" state waiting for response from the Manager. The Manager will process the association request and send the Association Response according with the APDU received. If the APDU corresponds to an already known MDS, the Manager will send an "Accepted" association response indicating that the configuration is already known, and then the Agent must transition to the Operating state. If the association request is accepted but the Manager does not recognize the MDS, it will send back an "accepted-unknown-config" association response asking to the Agent for the MDS configuration. If the association request is rejected, the Agent must transition to the Unassociated state and try again. A Manager's association response looks like the following.

```
0xE3 0x00 APDU CHOICE Type (AareApdu)
0x00 0x2C CHOICE.length = 44
0x00 0x03 result = accepted-unknown-config
0x50 0x79 data-proto-id = 20601
0x00 0x26 data-proto-info length = 38
0x80 0x00 0x00 0x00 protocolVersion
0x80 0x00 encoding rules = MDER
0x80 0x00 0x00 0x00 nomenclatureVersion
0x00 0x00 0x00 0x00 functionalUnits
0x80 0x00 0x00 0x00 systemType = sys-type-manager
0x00 0x08 system-id length = 8 and value
0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88
0x00 0x00 manager's response to config-id is always 0
0x00 0x00 0x00 0x00 manager's response to data-req-mode-capab is always 0
0x00 0x00 0x00 0x00 optionList.count = 0 | optionList.length = 0
```

Either if the Agent receives an accepted or accepted-unknown-config association response, the Agent must transition to the "Associated" state. In this case, the Manager accepted the association request, but it did not recognize the MDS returning an accepted-unknown-config association response. As a result of this, Agent must send a Configuration Report like the following.

```
/* configuration event report */
uint_8 USB_CONST PHD_OXI_CNFG_EVT_RPT[PHD_OXI_CNFG_EVT_RPT_SIZE] = {
0xE7, 0x00,                        /* APDU CHOICE Type (PrstApdu) */
0x00, 0x70,                        /* CHOICE.length = 112 */
0x00, 0x6E,                        /* OCTET STRING.length = 110 */
0x00, 0x02,                        /* invoke-id (differentiates this from other
outstanding messages) */
0x01, 0x01,                        /* CHOICE(Remote Operation Invoke | Confirmed Event
Report) */
0x00, 0x68,                        /* CHOICE.length = 104 */
0x00, 0x00,                        /* obj-handle = 0 (MDS object) */
0xFF, 0xFF, 0xFF, 0xFF,            /* event-time = 0xFFFFFFFF */
0x0D, 0x1C,                        /* event-type = MDC_NOTI_CONFIG */
0x00, 0x5E,                        /* event-info.length = 94 (start of ConfigReport) */
0x40, 0x00,                        /* config-report-id */
0x00, 0x02,                        /* config-obj-list.count = 2 Measurement objects will
be "announced" */
0x00, 0x58,                        /* config-obj-list.length = 88 */
0x00, 0x06,                        /* obj-class = MDC_MOC_VMO_METRIC_NU */
0x00, 0x01,                        /* obj-handle = 1 (.. 1st Measurement is SpO2) */
0x00, 0x04,                        /* attributes.count = 4 */
0x00, 0x24,                        /* attributes.length = 36 */
0x09, 0x2F,                        /* attribute-id = MDC_ATTR_ID_TYPE */
0x00, 0x04,                        /* attribute-value.length = 4 */
0x00, 0x02, 0x4B, 0xB8,            /* MDC_PART_SCADA | MDC_PULS_OXIM_SAT_O2 */
0x0A, 0x46,                        /* attribute-id = MDC_ATTR_METRIC_SPEC_SMALL */
0x00, 0x02,                        /* attribute-value.length = 2 */
0x40, 0xC0,                        /* avail-stored-data, acc-manager-init, acc-agent-
init, measured */
0x09, 0x96,                        /* attribute-id = MDC_ATTR_UNIT_CODE */
0x00, 0x02,                        /* attribute-value.length = 2 */
0x02, 0x20,                        /* MDC_DIM_PERCENT */
0x0A, 0x55,                        /* attribute-id = MDC_ATTR_ATTRIBUTE_VAL_MAP */
0x00, 0x0C,                        /* attribute-value.length = 12 */
0x00, 0x02,                     /* AttrValMap.count = 2 */
0x00, 0x08,                     /* AttrValMap.length = 8*/
0x0A, 0x4C, 0x00, 0x02,         /* MDC_ATTR_NU_VAL_OBS_BASIC | value length = 2 */
0x09, 0x90, 0x00, 0x08,         /* MDC_ATTR_TIME_STAMP_ABS | value length = 8 */
0x00, 0x06,                     /* obj-class = MDC_MOC_VMO_METRIC_NU */
0x00, 0x02,                     /* obj-handle = 2 (..2nd Measurement is pulse rate) */
0x00, 0x04,                     /* attributes.count = 4 */
0x00, 0x24,                     /* attributes.length = 36 */
0x09, 0x2F,                     /* attribute-id = MDC_ATTR_ID_TYPE */
0x00, 0x04,                     /* attribute-value.length = 4 */
0x00, 0x02, 0x48, 0x1A,         /* MDC_PART_SCADA | MDC_PULS_OXIM_PULS_RATE */
0x0A, 0x46,                     /* attribute-id = MDC_ATTR_METRIC_SPEC_SMALL */
0x00, 0x02,                     /* attribute-value.length = 2 */
0x40, 0xC0,                     /* avail-stored-data, acc-manager-init, acc-agent-
init, measured */
0x09, 0x96,                     /* attribute-id = MDC_ATTR_UNIT_CODE */
0x00, 0x02,                     /* attribute-value.length = 2 */
0x0A, 0xA0,                     /* MDC_DIM_BEAT_PER_MIN */
0x0A, 0x55,                     /* attribute-id = MDC_ATTR_ATTRIBUTE_VAL_MAP */
0x00, 0x0C,                     /* attribute-value.length = 12 */
0x00, 0x02,                     /* AttrValMap.count = 2 */
0x00, 0x08,                     /* AttrValMap.length = 8 */
0x0A, 0x4C, 0x00, 0x02,         /* MDC_ATTR_NU_VAL_OBS_BASIC, 2 */
0x09, 0x90, 0x00, 0x08          /* MDC_ATTR_TIME_STAMP_ABS, 8 */
};
```

This configuration report corresponds to the pulse oximeter device. Here the Agent indicates that it will send two numeric objects (all the possible objects are described in the ISO/IEEE 11073-20601:2010 document in the chapter 6: Personal health device DIM). The first numeric object corresponds to the oxygen saturation (SpO2) measurement. The second numeric object corresponds to the pulse rate measurement.

Once the configuration report has been sent, the Manager must respond indicating whether the reported configuration can be used or not. If the reported configuration can be used, the Agent must transition to the operating state. If the reported configuration is not supported by the manager, the Agent must try again using a different configuration that is supported by the Manager. A Manager’s response will look like the following.

```
0xE7 0x00 APDU CHOICE Type (PrstApdu)
0x00 0x16 CHOICE.length = 22
0x00 0x14 OCTET STRING.length = 20
0x43 0x21 invoke-id = 0x4321 (start of DataApdu. MDER encoded.)
0x02 0x01 CHOICE (Remote Operation Response | Confirmed Event Report)
0x00 0x0E CHOICE.length = 14
0x00 0x00 obj-handle = 0 (MDS object)
0x00 0x00 0x00 0x00 currentTime = 0
0x0D 0x1Cevent-type = MDC_NOTI_CONFIG
0x00 0x04 event-reply-info.length = 4
0x40 0x00 ConfigReportRsp.config-report-id = 0x4000
0x00 0x00 ConfigReportRsp.config-result = accepted-config
```

In this case, the Manager reported that configuration has been accepted and Agent must transition to the operating state.

As mentioned before, if the Agent receives either an accepted or accepted-unknown-config association response, the Agent must transition to the associated state. Once on the associated state, the Manager can use the “Get” service at any time to request the MDS attributes. The MDS attributes contain information about the MDS object like the kind of device (for example, glucose meter, thermometer, blood pressure monitor and others), company name, and device model among others. A Get all MDS attributes request looks like the following.

```
0xE7 0x00 APDU CHOICE Type (PrstApdu)
0x00 0x0E CHOICE.length = 14
0x00 0x0C OCTET STRING.length = 12
0x34 0x56 invoke-id = 0x3456 (start of DataApdu. MDER encoded.)
0x01 0x03 CHOICE (Remote Operation Invoke | Get)
0x00 0x06 CHOICE.length = 6
0x00 0x00 handle = 0 (MDS object)
0x00 0x00 attribute-id-list.count = 0 (all attributes)
0x00 0x00 attribute-id-list.length = 0
```

If a get all MDS attributes request is received, the Agent must respond with its attributes. Following example shows the response of the Get attributes command that the Agent sends to the Manager.

```
/* response to get attributes command */
uint_8 USB_CONST PHD_OXI_DIM_GET_RSP[PHD_OXI_DIM_GET_RSP_SIZE] = {
0xE7, 0x00,                     /* APDU CHOICE Type (PrstApdu) */
0x00, 0x6F,                     /* CHOICE.length = 111 */
0x00, 0x6D,                     /* OCTET STRING.length = 109 */
0x00, 0x02,                     /* invoke-id =0x0002 (mirrored from request)*/
0x02, 0x03,                     /* CHOICE (Remote Operation Response | Get)*/
0x00, 0x67,                     /* CHOICE.length = 103                   */
0x00, 0x00,                     /* handle = 0 (MDS object)              */
0x00, 0x06,                     /* attribute-list.count = 6             */
0x00, 0x61,                     /* attribute-list.length = 97            */
0x0A, 0x5A,                     /* attribute id=MDC_ATTR_SYS_TYPE_SPEC_LIST */
0x00, 0x08,                     /* attribute-value.length = 8            */
0x00, 0x01,                     /* TypeVerList count = 1                 */
0x00, 0x04,                     /* TypeVerList length = 4                */
0x10, 0x04,                     /* type = MDC_DEV_SPEC_PROFILE_PULS_OXIM */
0x00, 0x01,                     /* version=ver 1 of the specialization  */
0x09, 0x28,                     /* attribute-id = MDC_ATTR_ID_MODEL    */
0x00, 0x1B,                     /* attribute-value.length = 27        */
0x00, 0x0A, 0x46, 0x72,         /* string length = 10 | Freescale(space)  */
0x65, 0x65, 0x73, 0x63,
0x61, 0x6C, 0x65, 0x20,
0x00, 0x0D, 'M', 'E',         /* string length = 13 | MED-SPO2 PHDC    */
'D', '-', 'S', 'P',
'O', '2', ' ', 'P',
'H', 'D', 'C',
0x09, 0x84,                     /* attribute-id = MDC_ATTR_SYS_ID        */
0x00, 0x0A,                     /* attribute-value.length = 10           */
0x00, 0x08, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
                                /* octet string length = 8 | EUI-64  */
0x0a, 0x44,                     /* attribute-id = MDC_ATTR_DEV_CONFIG_ID   */
0x00, 0x02,                     /* attribute-value.length = 2            */
0x40, 0x04,                     /* dev-config-id = 16384
                                   (extended-config-start)*/
0x09, 0x2D,                     /* attribute-id = MDC_ATTR_ID_PROD_SPECN */
0x00, 0x12,                     /* attribute-value.length = 18          */
0x00, 0x01,                     /* ProductionSpec.count = 1             */
0x00, 0x0E,                     /* ProductionSpec.length = 14            */
0x00, 0x01,                     /* ProdSpecEntry.spec-type=1(serial-number)*/
0x00, 0x00,                     /* ProdSpecEntry.component-id = 0     */
0x00, 0x08, 0x44, 0x45,         /* string length = 8 |
                                   prodSpecEntry.prod-spec = DE124567 */

0x31, 0x32, 0x34, 0x35,
0x36, 0x37,
0x09, 0x87,                     /* attribute-id =MDC_ATTR_TIME_ABS */
0x00, 0x08,                     /* attribute-value.length = 8  */
0x20, 0x09, 0x06, 0x12,         /* Absolute-Time-Stamp=2009-06-12T12:05:0000*/
0x12, 0x05, 0x00, 0x00
};
```

In this example, the Agent describes it’s MDS as a pulse oximeter, the company name is “Freescale” and the device model is “MED-SPO2 PHDC”.

Once the Agent is in the Operating state, it can start reporting measurements to the Manager. Measurements must be sent using fixed reports. These reports must contain the measurements organized according with the MDS configuration report sent previously. For example, in our configuration report, the Agent indicated to the Manager that it will send two numeric measurements, a SpO2 value and a pulse rate value. Our MDS object result as follows:

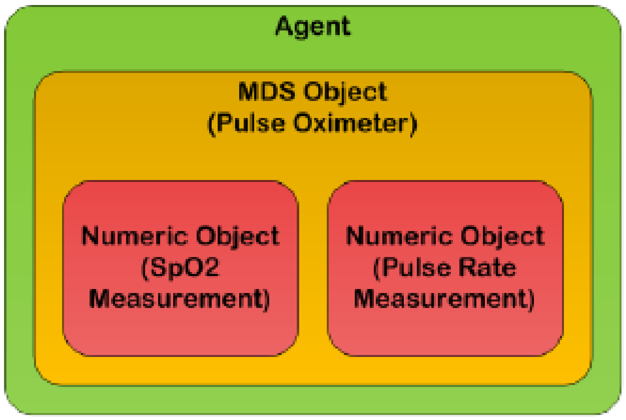


Figure 4. MED-SPO2 Agent representation

```
/* measurements to send */
uint_8  USB_CONST PHD_OXI_DIM_DATA_TX[PHD_OXI_DIM_DATA_TX_SIZE] = {
0xE7, 0x00, /*APDU CHOICE Type (PrstApdu)*/
0x00, 0x36, /*CHOICE.length = 54*/
0x00, 0x34, /*OCTET STRING.length = 52*/
0x12, 0x36, /*invoke-id = 0x1236*/
0x01, 0x01, /*CHOICE(Remote Operation Invoke | Confirmed Event Report)*/
0x00, 0x2E, /*CHOICE.length = 46*/
0x00, 0x00, /*obj-handle = 0 (MDS object)*/
0x00, 0x00, 0x00, 0x00, /*event-time = 0*/
0x0D, 0x1D, /*event-type = MDC_NOTI_SCAN_REPORT_FIXED*/
0x00, 0x24, /*event-info.length = 36*/
0xF0, 0x00, /*ScanReportInfoFixed.data-req-id = 0xF000*/
0x00, 0x00, /*ScanReportInfoFixed.scan-report-no = 0*/
0x00, 0x02,/*ScanReportInfoFixed.obs-scan-fixed.count = 2*/
0x00, 0x1C, /*ScanReportInfoFixed.obs-scan-fixed.length = 28*/
0x00, 0x01, /*ScanReportInfoFixed.obs-scan-fixed.value[0].obj-handle = 1*/
0x00, 0x0A, /*ScanReportInfoFixed.obs-scan-fixed.value[0]. obs-val-data.length
              = 10*/
0x00, 0x61, /*Simple-Nu-Observed-Value = 97% SpO2*/
0x20, 0x0B, 0x09, 0x23, /*Absolute-Time-Stamp = 2011-09-23T10:05:0000*/
0x0A, 0x05, 0x00, 0x00,
0x00, 0x02, /* ScanReportInfoFixed.obs-scan-fixed.value[1].obj-handle = 2*/
0x00, 0x0A, /* ScanReportInfoFixed.obs-scan-fixed.value[1]. obs-val-data.length
               = 10*/
0x00, 0x4E, /* Simple-Nu-Observed-Value = 78 BPM*/
0x20, 0x0B, 0x09, 0x23, /*Absolute-Time-Stamp = 2011-09-23T10:05:0000*/
0x0A, 0x05, 0x00, 0x00
};
```

In this APDU, the Agent reported two numeric objects, a 97 and a 78. The 97 is identified as the object handle 1 so the Manager can know that this measurement correspond to the SpO2. The same with the 78, which was reported as the object handle 2 so the Manager knows that this measurement corresponds to the pulse rate. A time stamp for each one of the measurements has been also sent as defined in the MDS configuration report.

3.3.2 Application Execution in the Microcontroller

The application execution in the microcontroller starts when the function TestApp_Task is called. This function is executed in an infinite loop and is constantly checking the status of the Agent's state machine.

The function TestApp_Task contains a small state machine that handles the status of the application. In a first instance, if the device is successfully enumerated as a PHD, the "event" variable is APP_PHD_INITIALIZED. The device first starts a timer, giving time to the user to select the MDS object they want for the association in case the Agent have more than one MDS object. After the timer finishes its count, the "event" variable is APP_PHD_SELECT_TIMER_OFF. Into this case statement, the PHD_Connect_To_Manager function is called. This function sends the Association Request defined in the file phd_device_spec.c and starts the association process described before. All the association process is handled automatically with the functions on the file phd_com_model.c and it takes all the required APDUs previously defined in the file phd_device_spec.c to complete the association. This helps developers to focus on their application forgetting all the logistics related with the PHD communication.

The SpO2_PeriodicTask function is called periodically into the TestApp_Task function. This function handles the pulse oximeter itself. It controls the required peripherals for the MED-SPO2 board handling and gets the SpO2 and pulse rate measurements. More information about the behaviour of this function can be found in the application note AN4327 Pulse Oximeter Fundamentals and Design. Following diagram represents the TestApp_Task function.

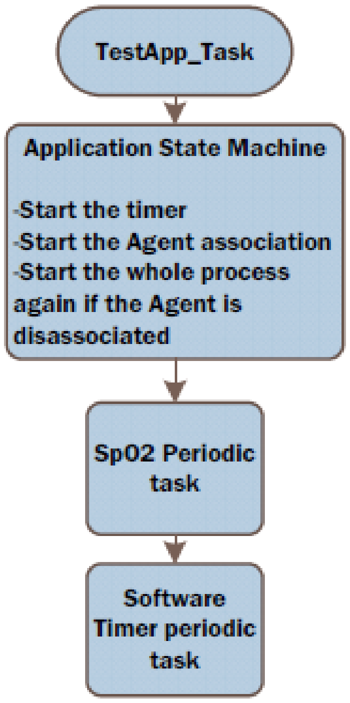

TestApp_Task
Application State Machine
-Start the timer
-Start the Agent association
-Start the whole process again if the Agent is disassociated
Sp02 Periodic task
Software Timer periodic task

Figure 5. TestApp_Task flow diagram

During the execution of the SpO2 periodic task, the SpO2 and pulse rate measurements are being constantly updated. In the SpO2 application initialization, a one second timer was created. This timer is activated as each time count is reached and restarted for another one second. When this timer is activated, it executes the function Send_PHDC_Measurements. This function counts the quantity of seconds elapsed, and when it detects that the quantity of second elapsed is the same as defined in SPO2_PHDC_UPDATE_PERIOD, it calls the function PHD_Send_Measurements_to_Manager.

The function PHD_Send_Measurements_to_Manager updates the measurement fixed report defined in the file phd_devicespec.c with the most recent measurements taken by the SpO2 periodic task function. Each 10 seconds a new set of measurements is sent and the Absolute Time Stamp is increased in one minute. The Manager then takes the measurements and shows them in its GUI.

4 Running the Demo

Following instructions will guide you in the assembling, software downloading and running of the demo.

4.1 Hardware Set

In order to assemble the demo, you will need the following parts.

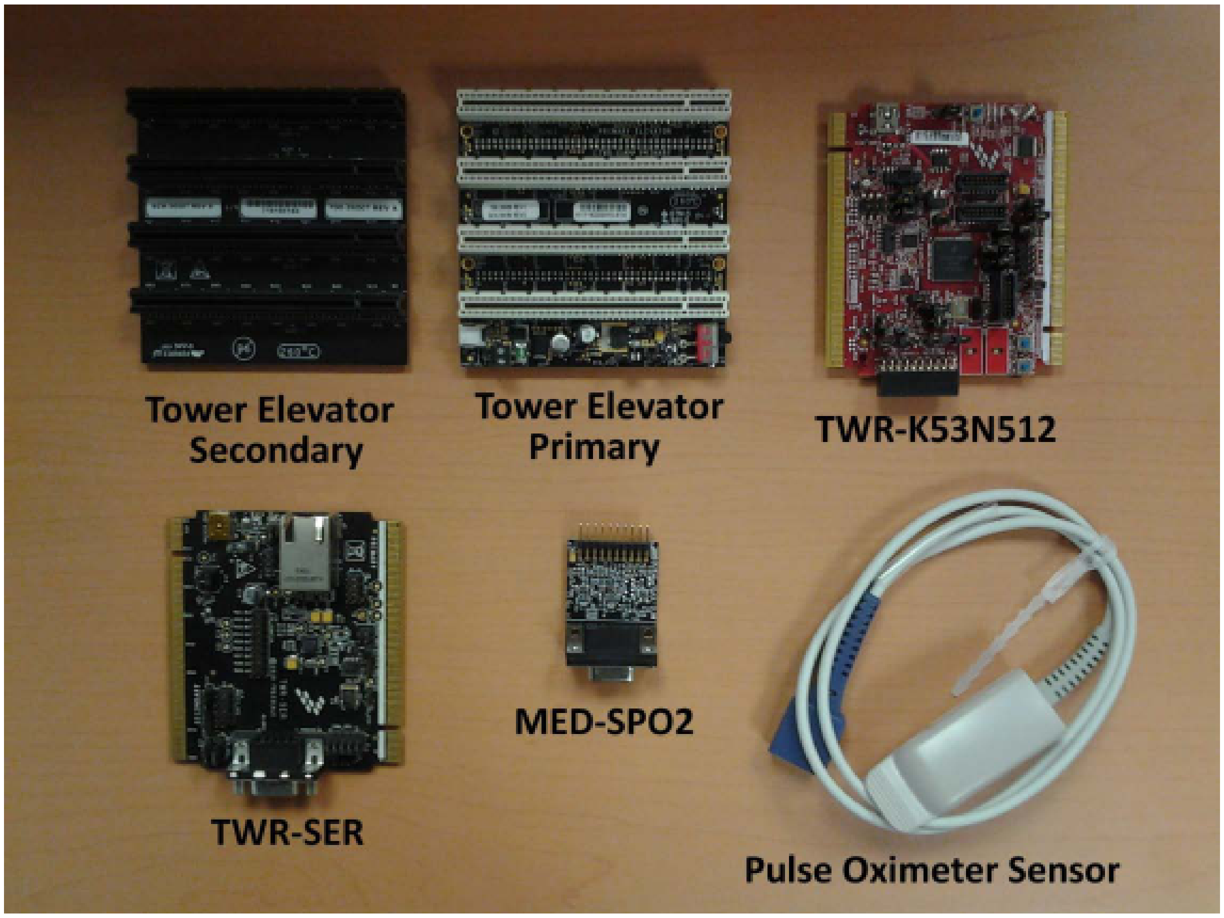


Figure 6. Required Title

TWR-K52N512 board and TWR-SER board requires to change the original jumper configuration in order to work. Make sure that the jumper configuration of these boards is like the one presented below.

Table 1. TWR-SER Jumper Configuration

Jumper	Position
J10	1-2
J16	3-4
J2	1-2

Table 2. TWR-K53N512 Jumper Configuration

Jumper	Position
J1	Open
J3	Open
J4	2-3

Table continues on the next page...

Table 2. TWR-K53N512 Jumper Configuration (continued)

Jumper	Position
J5	Open
J6	Connected
J7	Connected
J11	1-2
J12	Open
J14	Open
J15	Connected
J16	1-2
J17	Connected
J18	Connected
J20	Open
J21	Connected
J22	Open
J24	1-2
J25	Open
J26	Open
J28	Open
J29	Connected
J32	1-2
J33	1-2
J34	Open

4.1.1 Assembling the demo

Following steps will guide you on the demo assembling.

1. Take the TWR-K53N512 board and the Primary Elevator board. Connect the side of the TWR-K53N512 board marked as “Primary” to one of the slots on the Primary Elevator board.

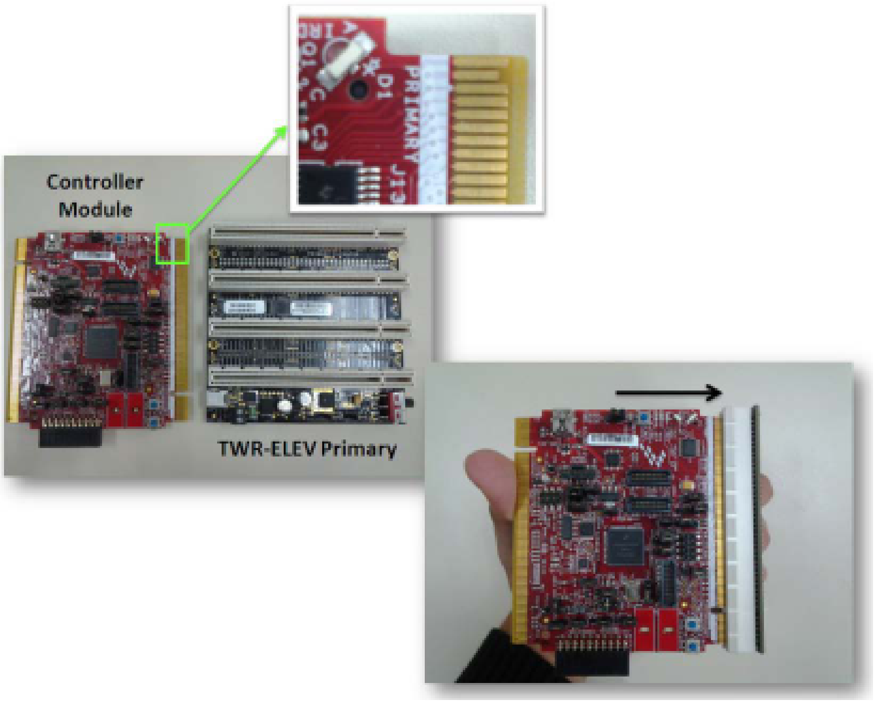


Figure 7. Assembling TWR-K53N512

2. Now take the TWR-SER board. Connect the side of the TWR-SER marked as primary to one of the slots on the Primary Elevator board.

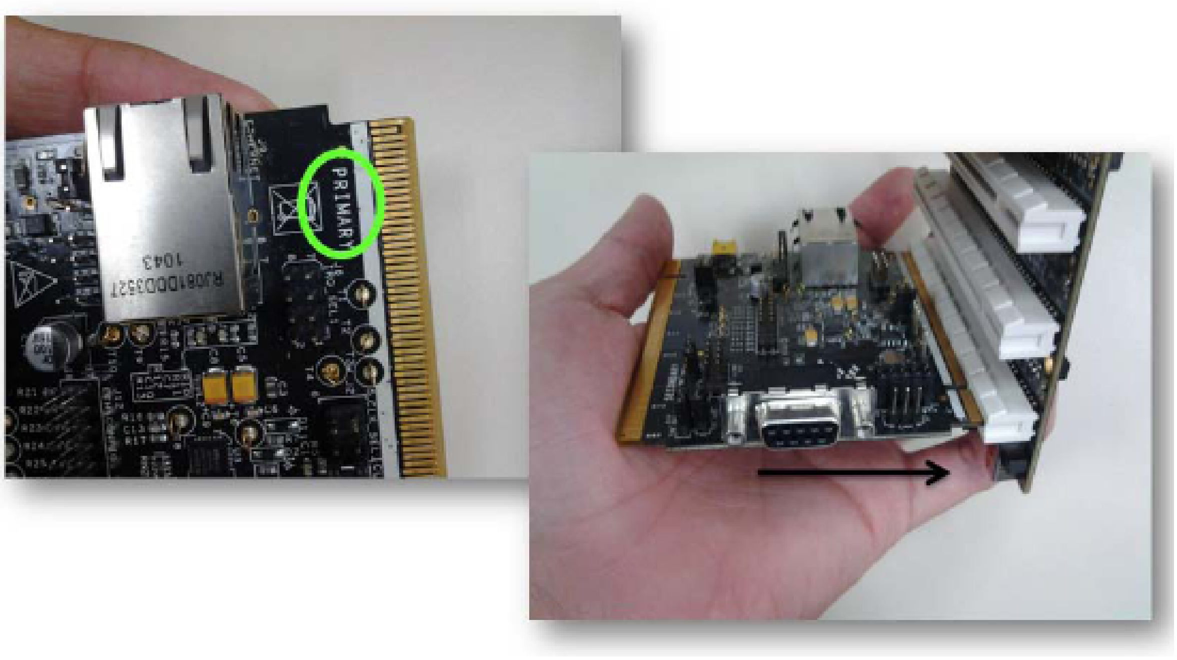


Figure 8. Assembling TWR-SER

3. Take the Secondary Elevator board. Connect the side of the TWR-SER and TWR-K53N512 boards marked as "Secondary" to the respective slot on the Secondary Elevator board.

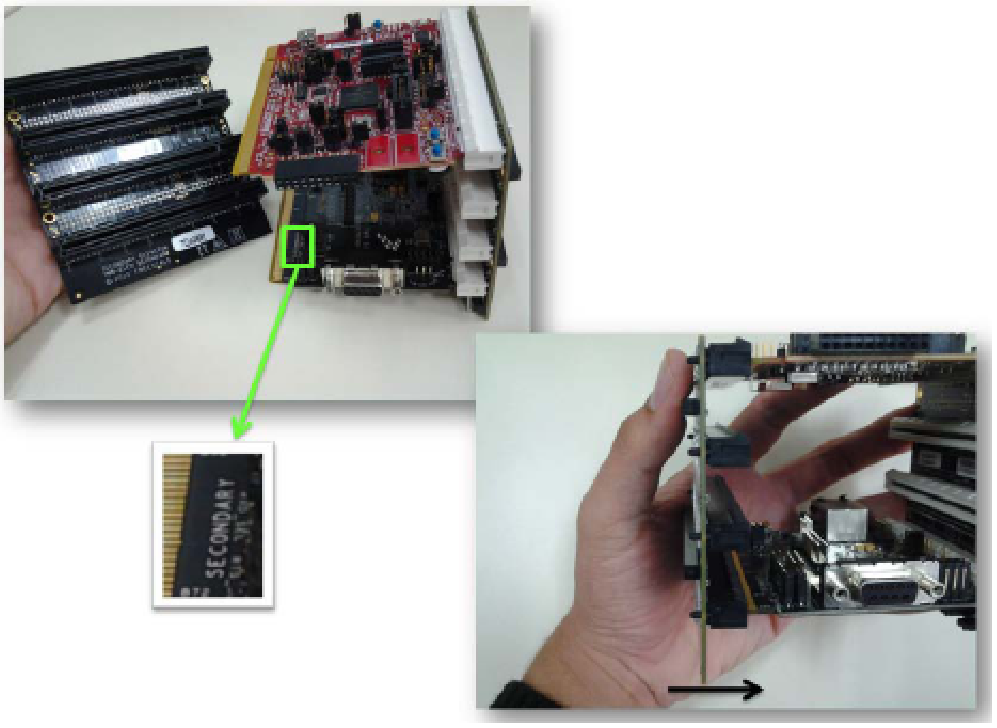


Figure 9. Assembling secondary Elevator

4. Take the MED-SPO2 board. Connect the pins on the MED-SPO2 board to the medical connector on the TWR-K53N512 board. Pin enumeration on the MED-SPO2 board must be mirrored with the pin enumeration on the TWR-K53N512 board (see the image below).

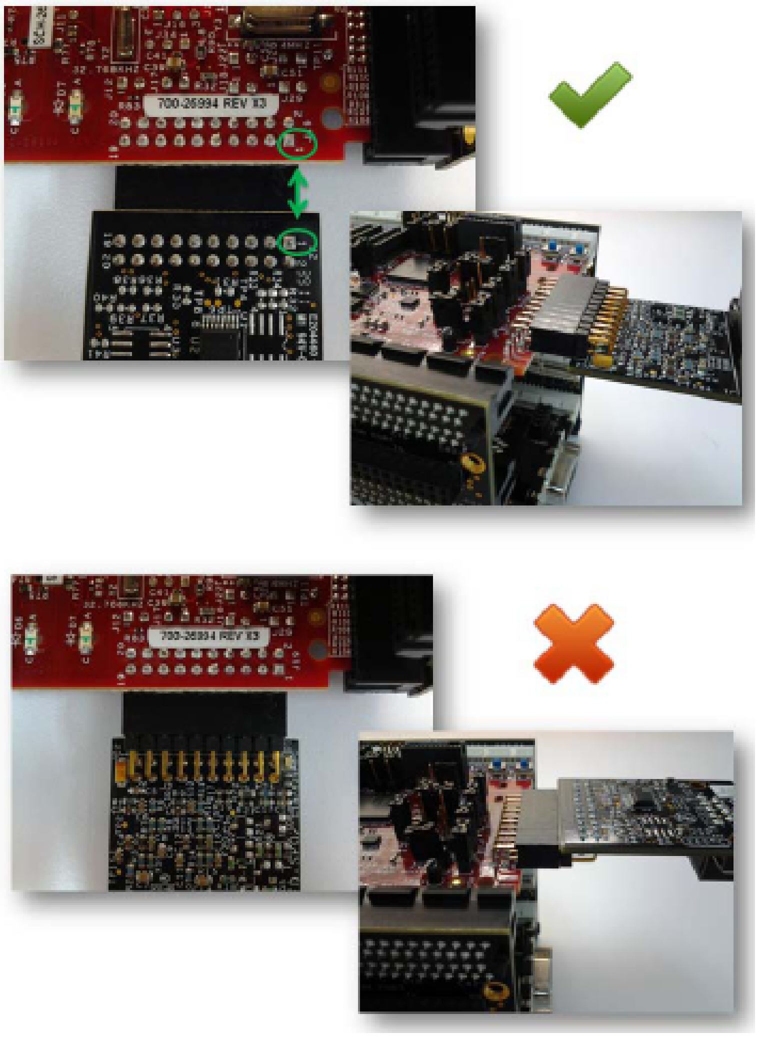


Figure 10. Analog front end placement

5. Connect the pulse oximeter sensor to the DB9 connector on the MED-SPO2 board.

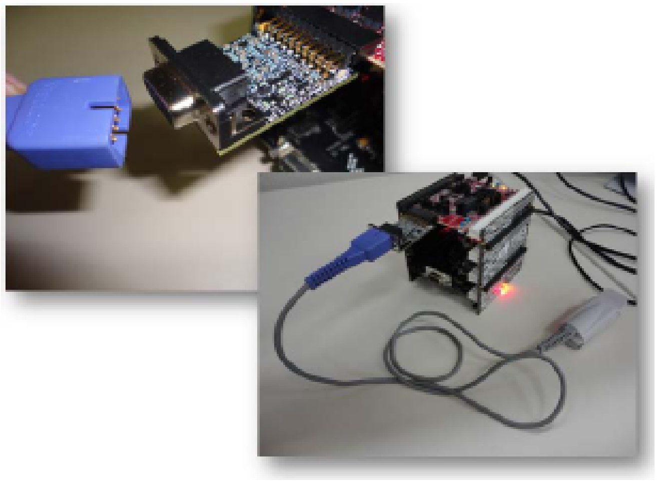


Figure 11. Pulse oximeter sensor placement

4.2 Demo Execution

1. Download and install HealthLink®. It can be found on the Lamprey Networks web page www.lnihealth.com .

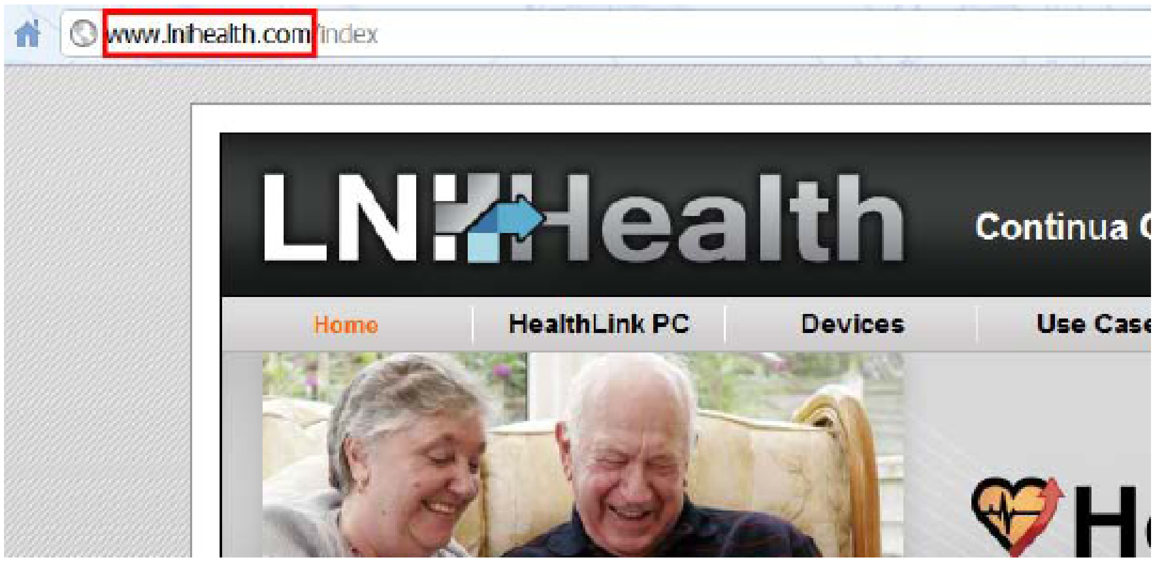


Figure 12. LNI Health web page

2. Connect an A to mini B USB cable from the computer to the TWR-SER USB port.

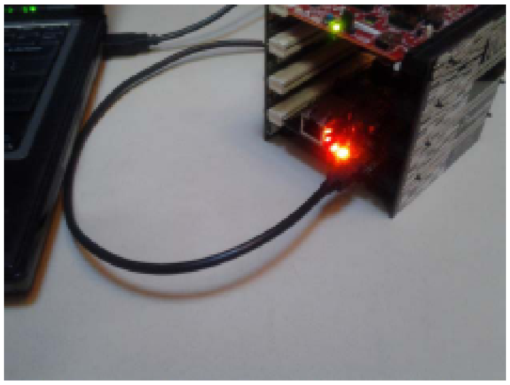
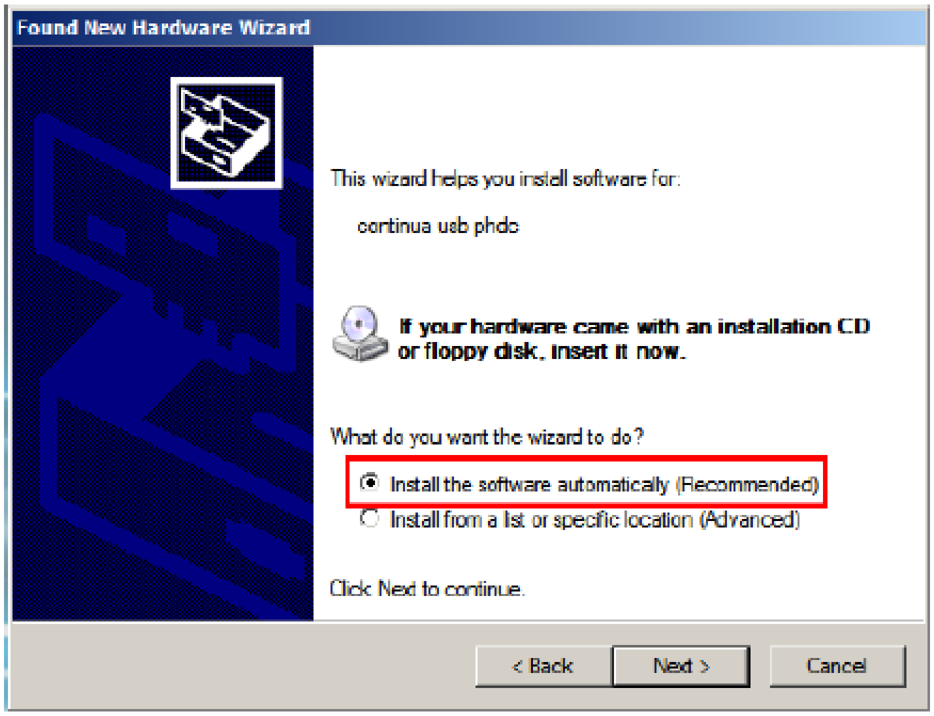


Figure 13. Connecting USB to TWR-SER

3. If a window asking for the USB PHDC drivers pops up, select the option “Install the drivers automatically”. Drivers are copied to the system folder during the HealthLink® program installation.



Found New Hardware Wizard

This wizard helps you install software for:

cortinua usb phdc

If your hardware came with an installation CD or floppy disk, insert it now.

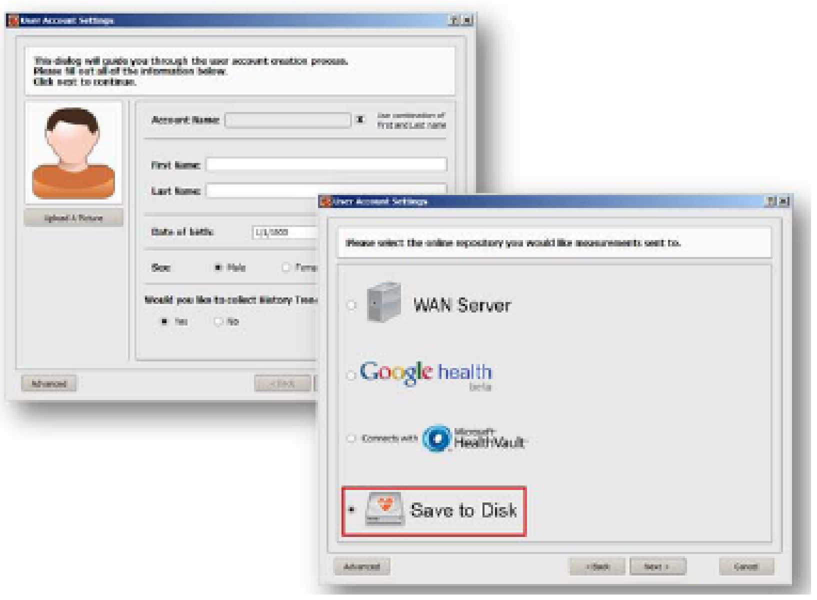
What do you want the wizard to do?

- [✓ Install the software automatically (Recommended)]
- Install from a list or specific location (Advanced)

Click Next to continue.

Figure 14. Installing the PHDC drivers

4. Execute the HealthLink® program. If it is the first time you use the program it will request you to create an account. Create a user account selecting your health data provider (i.e. Google Health, Microsoft HealthVault, etc...). If you don't have a health data provider you can use the option "Save to disk".



User Account Settings

This dialog will guide you through the user account creation process.
Please fill out all of the information below.
Click next to continue.

Account Name:

First Name:

Last Name:

Date of birth: 1/1/1900

Sex: [✓ Male] Female

Would you like to collect History Trend: [✓ Yes] No

User Account Settings

Please select the online repository you would like measurements sent to.

- WAN Server
- Google health beta
- Connects with Microsoft HealthVault
- [✓ Save to Disk]

Figure 15. Creating an account

5. Place the pulse oximeter sensor on the index finger as shown in the image below.

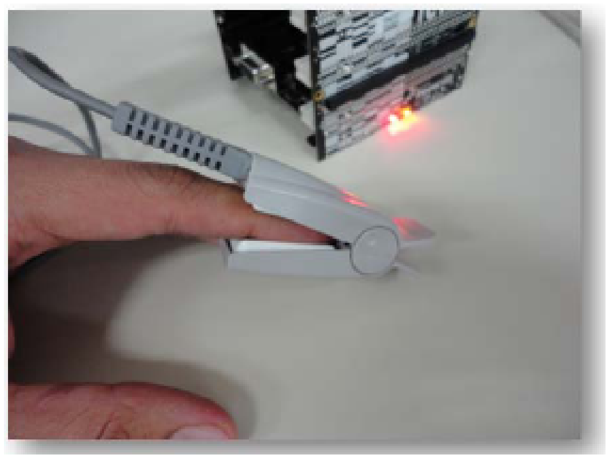
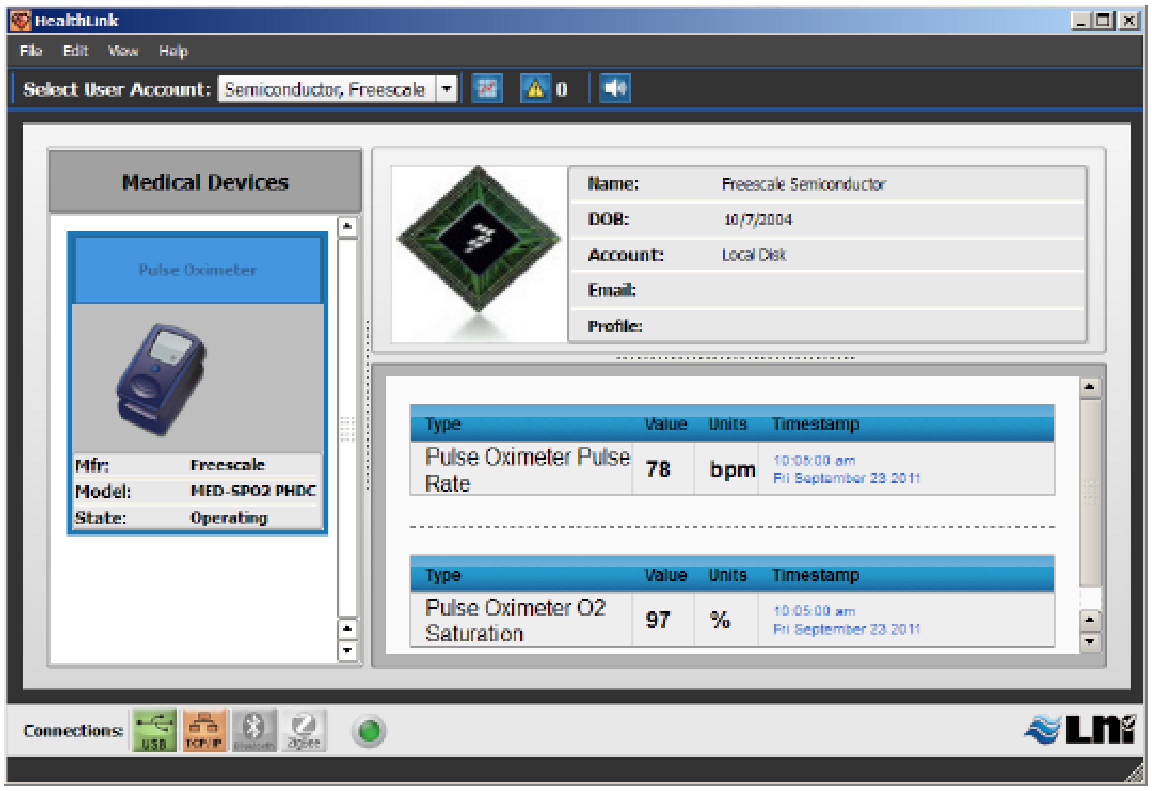


Figure 16. Finger sensor placement

6. When the account is active, the HealthLink® program will recognize the tower system as a pulse oximeter device. Measurements will be sent each ten seconds.



Select User Account: Semiconductor, Freescale

Medical Devices

Pulse Oximeter

Mfr:	Freescale
Model:	MED-SPO2 PHDC
State:	Operating

Name:	Freescale Semiconductor
DOB:	10/7/2004
Account:	Local Disk
Email:	
Profile:	

Type	Value	Units	Timestamp
Pulse Oximeter Pulse Rate	78	bpm	10:05:00 am Fri September 23 2011

Type	Value	Units	Timestamp
Pulse Oximeter O2 Saturation	97	%	10:05:00 am Fri September 23 2011

Figure 17. Demo running

5 References

- The development of the pulse oximeter is based on the application note "AN4327 Pulse Oximeter Fundamentals and Design"
- Software is based on the USB Stack with PHDC 3.0 which can be found on the Freescale web page https://www.freescale.com.

- The communication protocol is based in the standard ISO/IEEE 11073-20601 Personal Health Device Communications: Optimized Exchange Protocols
- The PHD pulse oximeter communication protocol implementation was developed accordingly with the IEEE 11073-10404 Personal Health Device Communications: Device Specialization-Pulse Oximeter
- This software was developed using IAR 6.3. It can be downloaded from the IAR web page https://www.iar.com
- The GUI used in the development of this demo is the HealthLink® GUI from Lamprey Networks and can be downloaded from the LNI web page https://www.lnihealth.com

6 Conclusions

The personal health care device class allows the same interoperability among portable medical devices. Freescale offers connectivity solutions that help developers in the creation of devices that are able to communicate with standards such as IEEE 11073-20601, making them a better choice in the market.

Document Number: AN4496
Rev. 0, 03/2012