Freescale Semiconductor
Application Note

Document Number:AN4644
Rev. 0, 01/2013

Remote Monitoring Solution Using MQX and Kinetis

by: Carlos Musich, Alí Piña, and Carlos Casillas

Contents

1 Introduction

Remote monitoring has become a need rather than an option in the embedded world. In fact, medical monitoring applications such as vital sign monitor are becoming very popular and demanded.

A vital sign monitor is a multi-parameter device that measures blood pressure, temperature, oxygen saturation, and heart electrical activity to give a clear view of patient information. This application note is intended to demonstrate the implementation of a Remote Medical Monitor System using K53, K60, MED-EKG, and Freescale MQX™ RTOS capabilities. The system consists of two parts:

- Medical Client: It is implemented in the TWR-K53N512-KIT development module and uses the MED-EKG board.
- Medical Server: The hardware used is TWR-K60N512-KIT, in addition to the TWR-LCD board.

The application source code described in this document can be found in the AN4644SW.zip file. For a full description of Freescale MQX RTOS, please visit **freescale.com/mqx.**

2 Overview

The objective of this application is to implement 4 clients that get a patient's heart signal and heart rate through a MED-EKG development board and send it over Ethernet to a server. It is possible to increase the number of clients depending on the requirements and hardware availability. The server graphs the heart signal and displays the heart rate in a TWR-LCD screen using eGUI. The server only displays the information of the client which is being selected in the server side. Figure 1 shows a high-level diagram of the entire application.

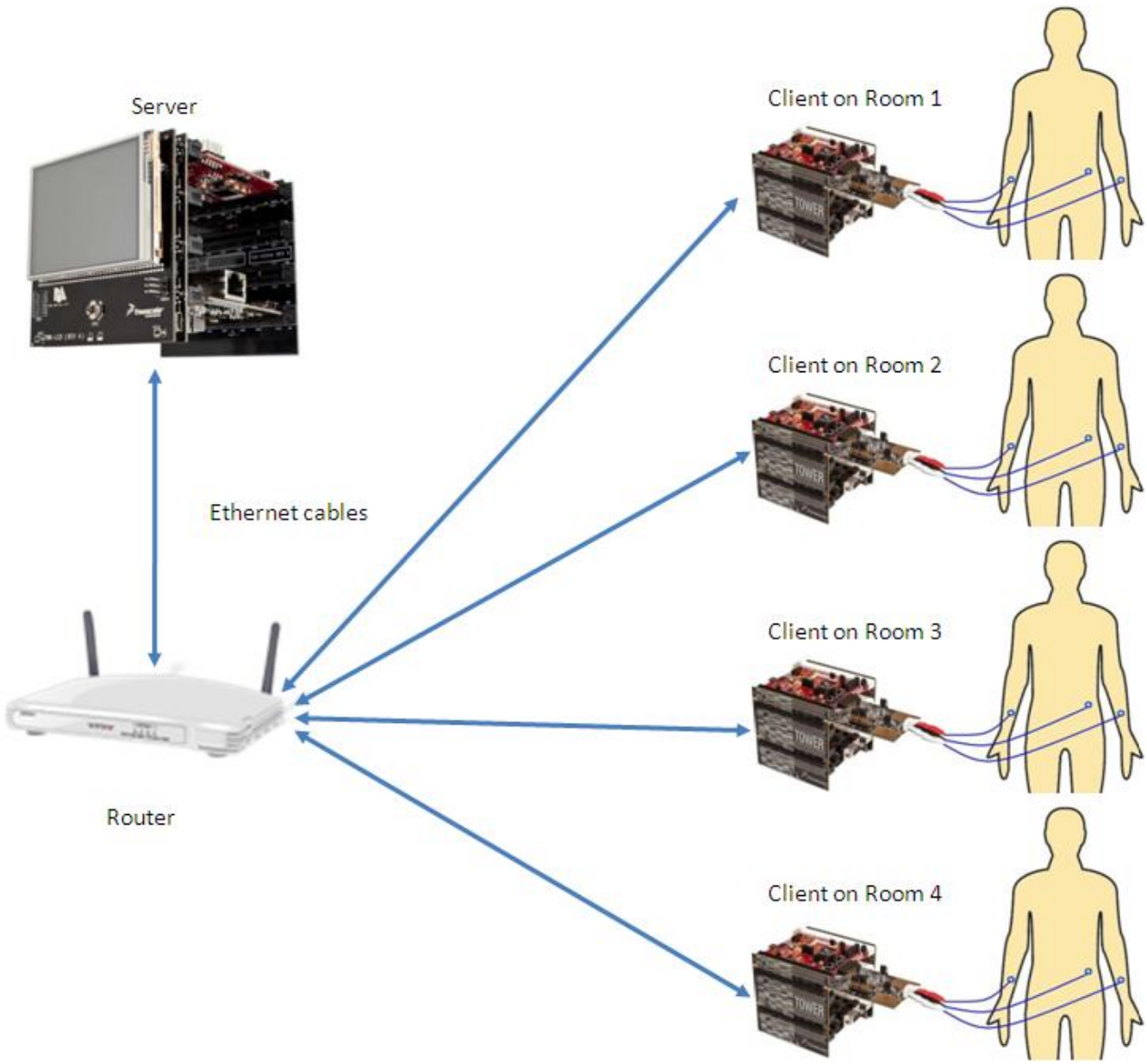

Figure 1. General block diagram

3 Client

This section explains the implementation of the Clients; in addition, it includes flowcharts of each task and/or process.

3.1 Main task

As shown in Figure 2, the Client's Main task is a simple task which can be explained by the following steps.

1. First, the Main task initializes the EKG task to start retrieving the patient's heart signal information.
2. Then, it initializes RTCS to communicate with the server through Ethernet.
3. Finally, it starts the Discovery task which is explained in the following section.

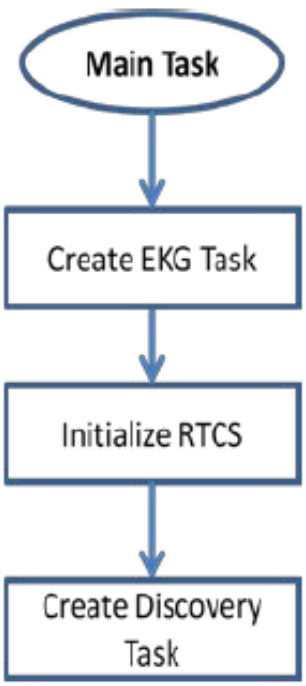

Figure 2. Client Main task flowchart

3.2 Discovery task

As the networking configuration is set to obtain IP address dynamically, the clients are not able to know the server address when they are connected to the network. This task listens to broadcast messages sent by the server. These messages contain the server IP.

The whole task is an endless loop and can be summarized through the following steps.

1. First, it creates and binds a UDP socket.
   To create the socket, the instruction `sock = socket(AF_INET, SOCK_DGRAM, 0)` is used. The settings chosen to create the socket are shown below:
   - AF_INET: protocol family
   - SOCK_DGRAM: type of communication (UDP)
   - 0: specific protocol

To bind the socket, it is necessary to create a structure of type sockaddr_in. This time it was called local_sin and contains the following information.

```
local_sin.sin_family = AF_INET;
local_sin.sin_port = SERVER_BROADCAST_PORT;
local_sin.sin_addr.s_addr = INADDR_ANY;
```

In the code given above:

- `sin_family` indicates the protocol family.
- `sin_port` indicates the port number to be used. In this case, the port (SERVER_BRADCAST_PORT) is 1040.
- `s_addr` indicates that the socket binds to the local address.

2. Next, the socket is configured to listen to all ports with the function `RTCS_selectall(0)`. Zero as a parameter means that there is no timeout for the socket to receive data. The socket will keep listening for any message within the network. The function used to receive the messages is `recvfrom()` indicating the socket that was created, a buffer where the message is going to be received, and the sockaddr_in type structure that was created for this socket.

NOTE

For details about `sendto()` and `recvfrom()` functions, see MQX RTCS User's Guide located in the documentation folder in MQX installation path.

3. When a UDP message is received, the application searches in the buffer for the text "MedicalMonitorServer". If the string is found, the 8 characters next to the string are taken to be converted from ASCII to hexadecimal numbers in order to get the IP address. After the IP, the next character indicates the room number. The client will send its data only if the room number indicated in the server message matches with its own number. The room number is found in the buffer after the IP address.

Figure 3 describes Discovery task's flow.

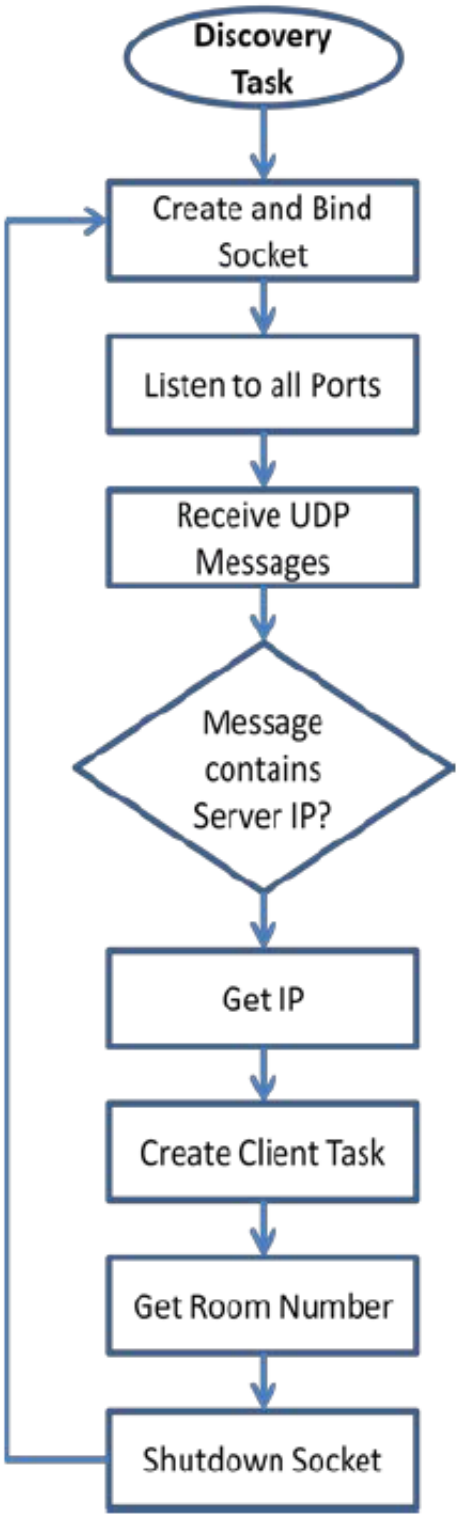

Figure 3. Discovery Task flowchart

3.3 Client task

As shown in Figure 4, once the client obtains the server IP, the Client task is created. The purpose of this task to send the data acquired by the EKG task to the server. The data is sent in a buffer with the information to print the EKG signal and the heart rate.

1. The task creates a socket and binds it. The socket options are the same as used in Discovery task. Then, the same sockaddr_in type structure that was created to bind the socket is configured to connect to port 1030 to a remote address, which is the server address obtained by Discovery task.

   ```
   addr.sin_port        = DESTPORT;
   addr.sin_addr.s_addr = remoteIP;
   ```

2. After configuring the socket, the task enters an endless loop which waits until the data buffer is ready to be sent. The instruction used to wait for such event is `_lwevent_wait_ticks()`.

   As the EKG Task uses a ping-pong buffer to send the monitor data, it is necessary to verify which of the buffers is ready. The function `_lwevent_get_signalled()` tells which event was executed; this way it is possible to know which buffer is ready.

   NOTE

   For details about `_lwevent_wait_ticks()` and `_lwevent_get_signalled()`, see MQX Reference Manual located in the documentation folder in MQX installation path.

3. The buffer is sent to the server using the function `sendto()` indicating the socket created in this task, and the sockaddr_in type structure which contains the server IP and the buffer with the data. Finally, the event is cleared in order to be ready for the next signal.

   NOTE

   For details about `sendto()` and `recvfrom()` functions, see MQX RTCS User's Guide located in the documentation folder in MQX installation path.

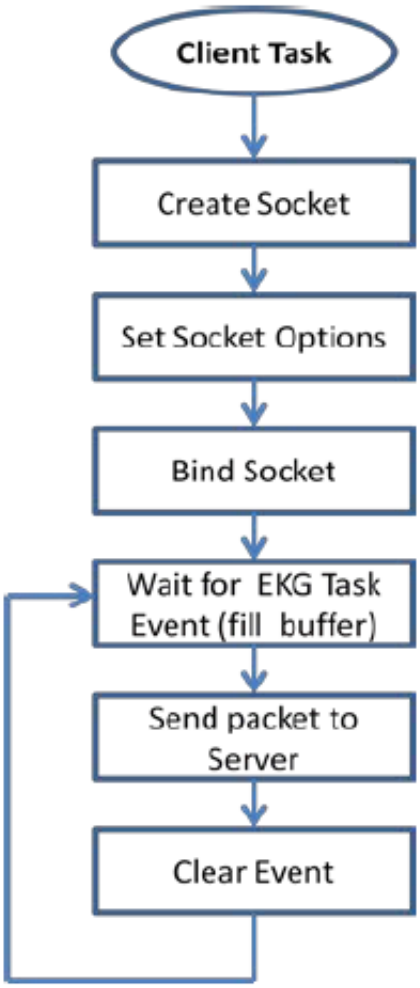

Figure 4. Client task flowchart

3.4 EKG task

The EKG task requires enabling the ADC0 driver of TWR-K53N512's BSP.To do this, open the user_config.h file included on the BSP project and locate the following macro: `#define BSPCFG_ENABLE_ADC`; set the value of the macro to 1, and recompile the BSP. The initialization of EKG task can be explained as follows. See Figure 6

- Calling the MQX's ADC driver initialization, setting 16-bit conversions with sampling period of 2 milliseconds, asserting an event when conversion completes.
- Enabling the Medical Connector power supply, for providing power to MED-EKG board.
- Initializing both transimpedance amplifiers (TRIAMPs) and one operational amplifier (OPAMP) in order to generate an Instrumentation Amplifier configuration, and then, using the second OPAMP as non-inverting amplifier with software-configurable gain.
- Creating an event that will be asserted when any of the buffers that contain Room ID, Averaged Heart Rate and the EKG samples become ready to be sent. The task "clientTask" will ask for this event, to know when to send a UDP packet.

Once the initialization has been completed, the EKG task enters into an infinite loop. On this loop, the following actions are performed:

1. It waits for the ADC conversion complete event. When it is asserted, the application goes to read the current ADC sample using MQX's ADC driver.

2. The new sample is sent to the FIR function execution, which returns the current filtered value, taking into account, the actual and “n” previous samples.
3. The implemented filter is 30th order band-pass Finite Impulse Response. As it is for filtering EKG signal, its pass band is from 0.1 Hz to 150 Hz.
4. The filter output signal is used to perform the heart rate calculation and automatic gain control. These processes will be explained in the following sections. Finally, the sample is stored on ping-ping buffers that will be sent by Ethernet task.

Figure 5 shows the block diagram of FIR sub-process, while Figure 6 shows the flowchart of EKG Task:

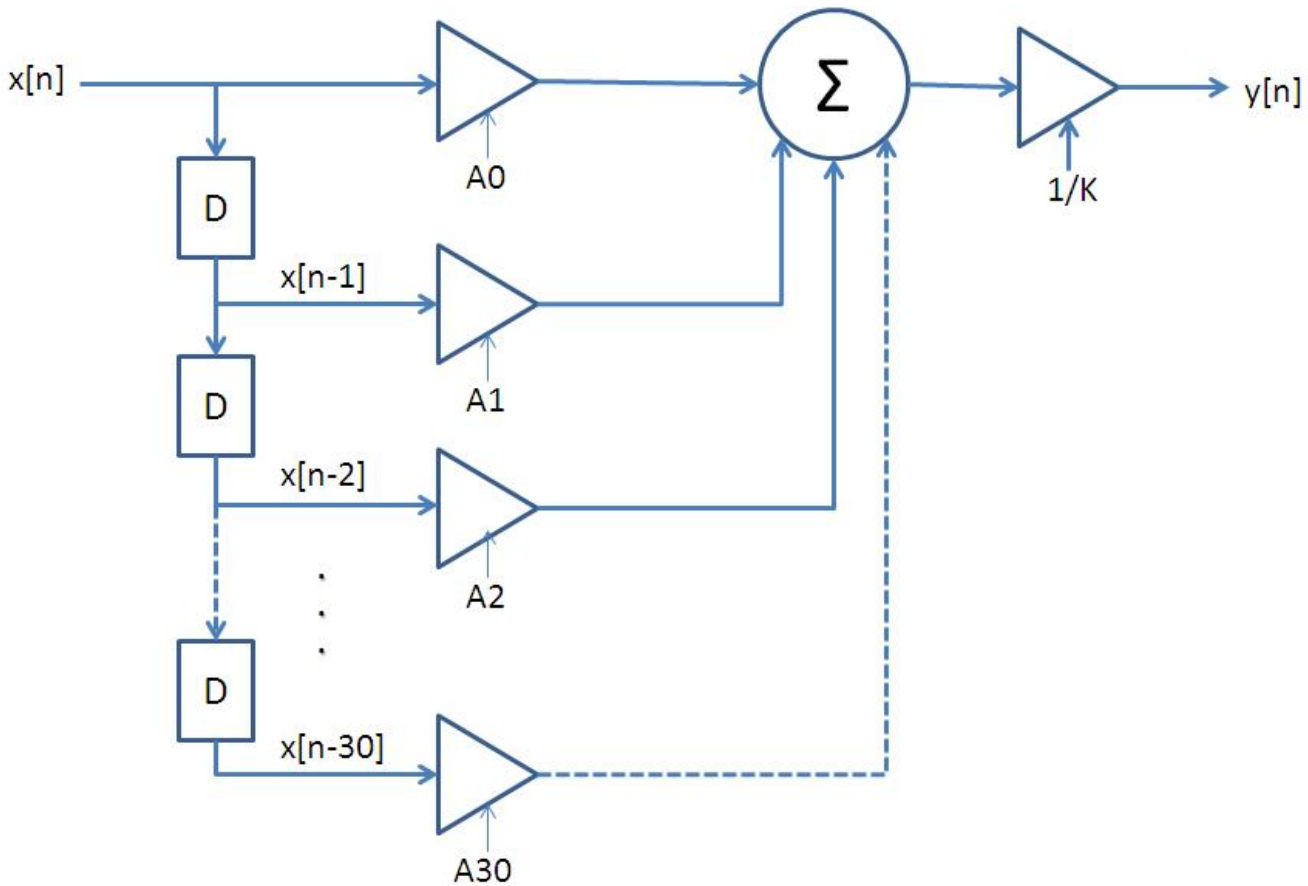

Figure 5. FIR filter sub-process block diagram

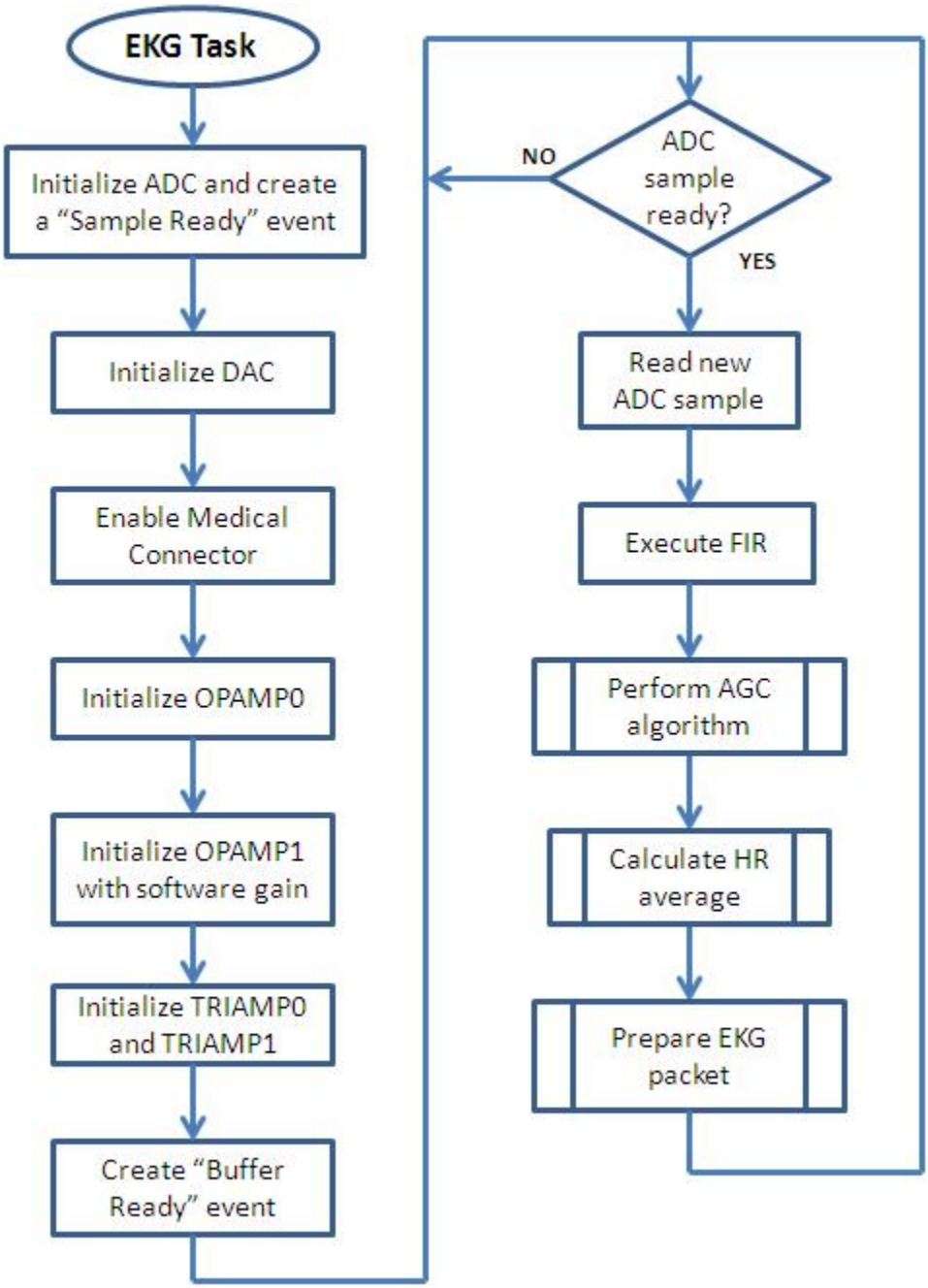

Figure 6. EKG Task flowchart

The Automatic gain control (AGC) algorithm involves constantly checking the amplitude of the EKG signal to change the gain of the internal amplifier in real time, if it is required, in order to maintain the signal between appropriate levels. The basic algorithm can be explained as follows.

1. There is a counter that is incremented each time that a sample of the EKG signal is taken. This counter serves to maintain a register of the time elapsed.
2. Considering a minimum heart rate of 40 bpm, a pulse must occur at least every 1.5 seconds. A window time of 1.2 seconds (50 bpm) is acceptable to detect a heartbeat since heart rate is generally higher than 40 bpm.
3. Within the time window, the application looks for the lowest and highest samples of the EKG signal by constantly updating two registers that contain the current minimum and maximum values. Once the time window ends, the difference between those values determines the maximum peak-to-peak amplitude of the signal during that period.

4. The software checks if the amplitude value is between the proper ranges; if doesn't match, the gain of the internal amplifier is changed either to increase or decrease it.

The flowchart of Automatic-Gain-Control sub-process is shown in Figure 7 :

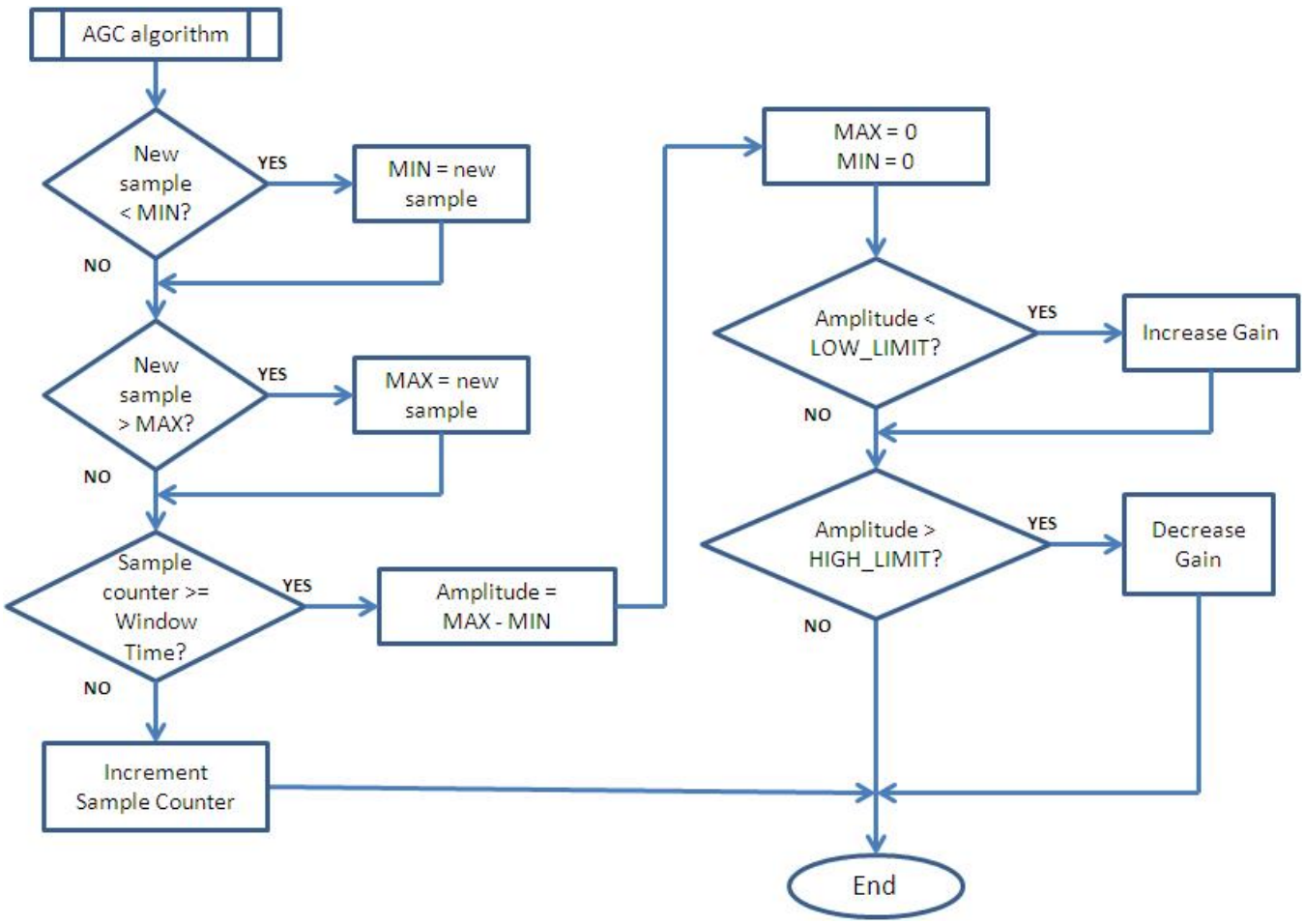

Figure 7. Automatic-Gain-Control sub-process flowchart

The Heart Rate Average calculation also uses a sample counter, but it is incremented after every 10 filtered samples. The process of Heart Average Rate calculation can be summarized as follows.

1. It waits for detecting a valid QRS complex pulse. It is validated using three samples. A valid QRS pulse is detected if:
   - the first sample is greater than the second sample,
   - the second sample is greater than the third sample,
   - the amplitude delta between first and third sample is greater than a defined threshold, and
   - the time delta between the actual pulse and the previous pulse is greater than a defined window of samples
2. When a valid QRS pulse is detected, its sample counter value is stored. When the next QRS pulse is detected, the period between pulses is calculated using the actual sample counter and the previously stored one. Then, the calculated period is stored into an array.
3. If the number of stored periods is equal to the value of `HR_PULSES_AVERAGE` macro, all the stored values are summed, and the result is divided by the value of `HR_PULSES_AVERAGE` macro.
4. Then, the averaged period value is used to calculate the number of beats per minute, and the averaged period value is cleaned.

The Heart-Rate Average sub-process flowchart is shown in Figure 8 :

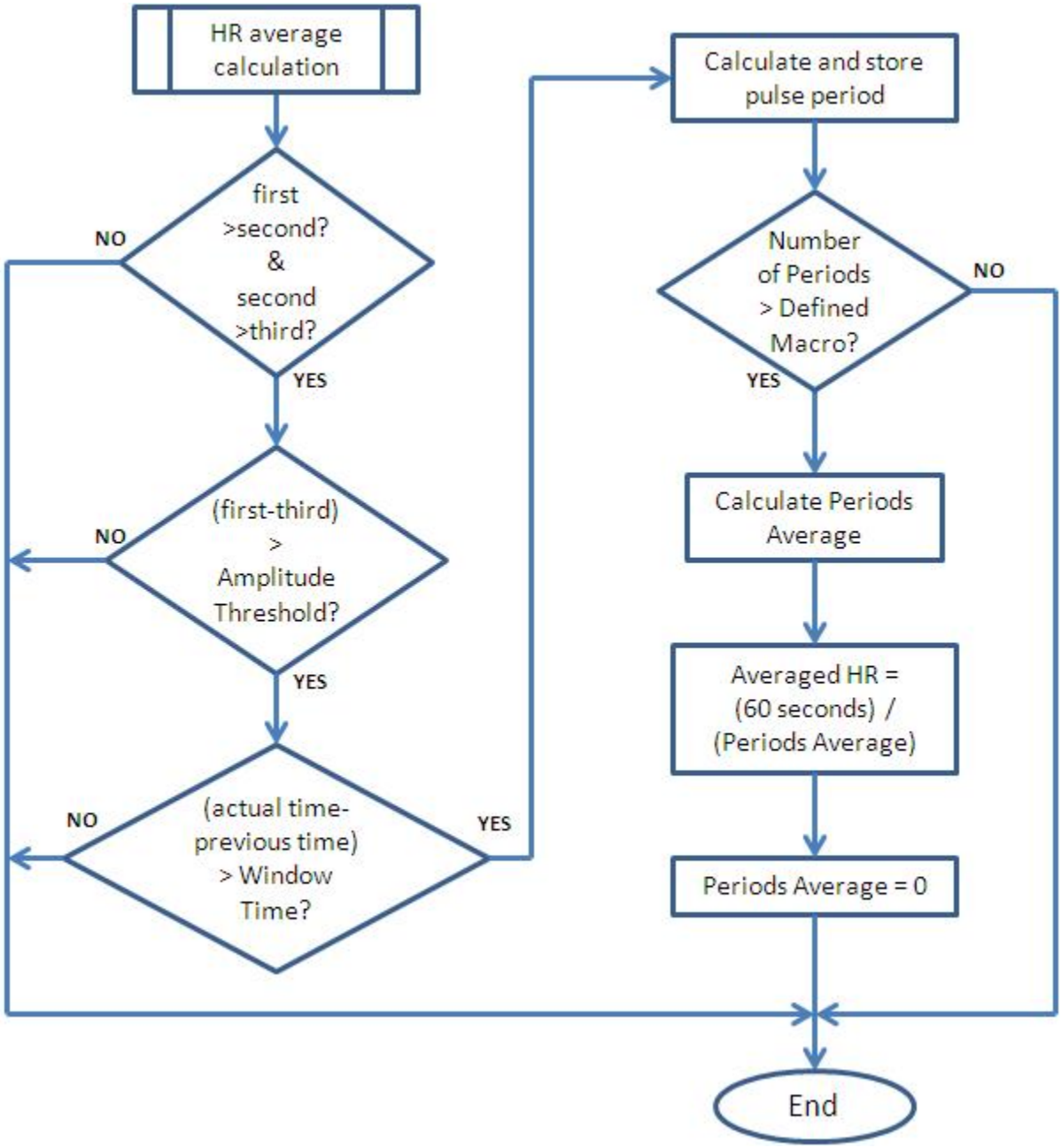

Figure 8. Calculate Heart-Rate Average sub-process flowchart

Finally, the function `EKG_prepare_packet` is responsible for preparing the even/odd buffers that will be sent by Ethernet tasks. This can be explained as follows.

1. If the variable that indicates which buffer is in use equals to ODD, the ODD buffer will be used; if the variable equals to EVEN, the EVEN buffer will be used.
2. The new incoming sample (that came from FIR output) is stored in the proper buffer, starting by position 2 and it is repeated until the sample position counter equals to `EKG_PACKET_SIZE`.
3. When sample position counter equals `EKG_PACKET_SIZE`, the position 0 is filled with `CLIENT_ID` value, and position 1 is filled with the actual heart rate average.
4. Finally, the application toggles the variable that determines which buffer is in use; then, the flag that indicates that a packet is ready takes a nonzero value for showing which buffer is ready and asserting the event that indicates that the mentioned buffer is ready to be sent by the other tasks.

The flow of Prepare EKG packet sub-process is shown in Figure 9 :

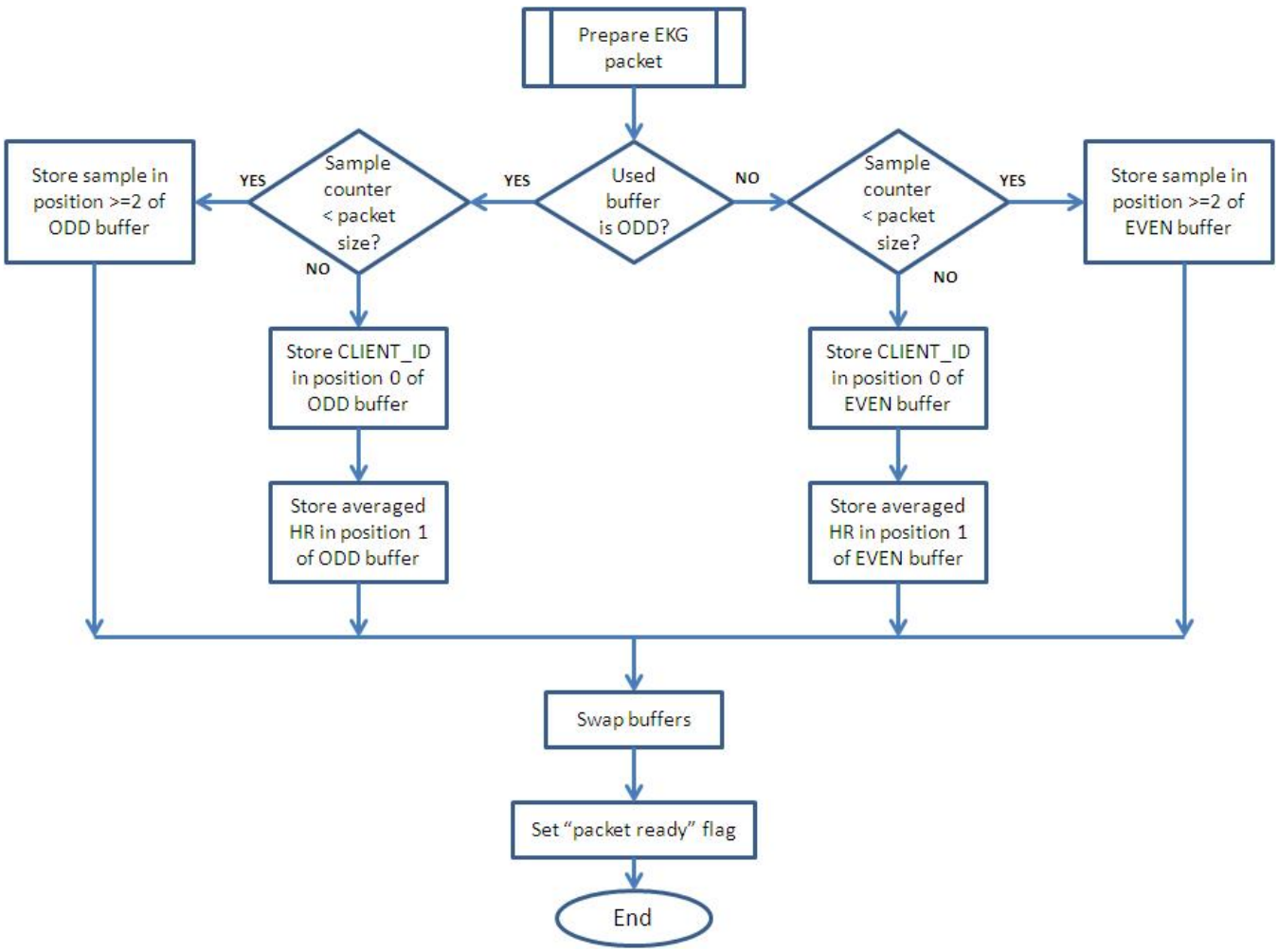

Figure 9. Prepare EKG packet sub-process flowchart

4 Server

The following table describes various server tasks.

Table 1. Description of Server tasks

Server tasks	Function
Main task	Configures the RTCS and creates the Broadcast task.
Broadcast task	Sends broadcast messages with its IP in order to be identified by any client that is connected to the network and creates Server task.
Server task	Receives UDP messages from Clients to be managed by LCD task.
LCD task	Installs the touch screen driver and creates the Time task.

Table continues on the next page...

Table 1. Description of Server tasks (continued)

Server tasks	Function
Time task	Enters into an infinite loop and gives a timer tick to eGUI library.

The architecture of each task/process included on the server (TWR-K60N512) is explained in the following sections.

4.1 Main task

Similar to the Client's Main task, the Server's Main task is also simple. It initializes the RTCS to start Ethernet communication and then it creates the Broadcast task. Figure 10 shows this behavior.

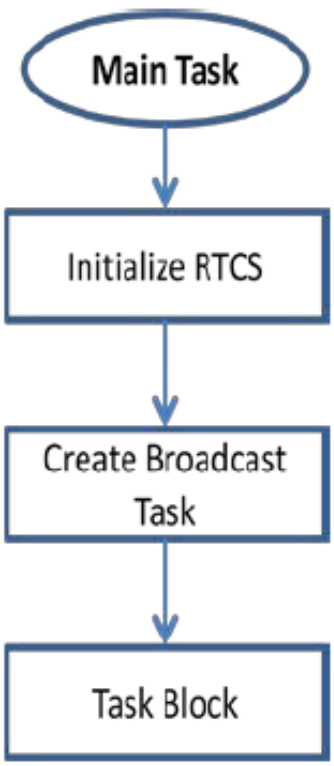

Figure 10. Server Main task flowchart

4.2 Broadcast task

As mentioned in Discovery task, the networking configuration is set to obtain IP address dynamically, therefore, the clients are not able to know the server address when they are connected to the network. This task is in charge of sending messages periodically with the server IP address. Then, clients will be able to discover the server address. Broadcast task process is shown in Figure 11.

1. This task creates the Server task and then enters into an endless loop. In this loop, UDP socket is created and bound. The process of creating and binding the socket is the same as in Discovery task.
2. To know the server IP address the following instruction is used:

```
ipcfg_get_ip(BSP_DEFAULT_ENET_DEVICE, &MyIP_data)
```

Server

3. Once the server IP is obtained, the function `snprintf()` is used to fill a buffer with the text “MedicalMonitorServer”, the IP address, and the room number to be displayed on the LCD.
4. The socket sends this buffer by UDP as a broadcast message using instruction `sendto()`.
5. Finally, the socket is closed and the infinite loop restarts.

NOTE

For details about the `ipcfg_get_ip()` function, see MQX RTCS User’s Guide located in the documentation folder in MQX installation path.

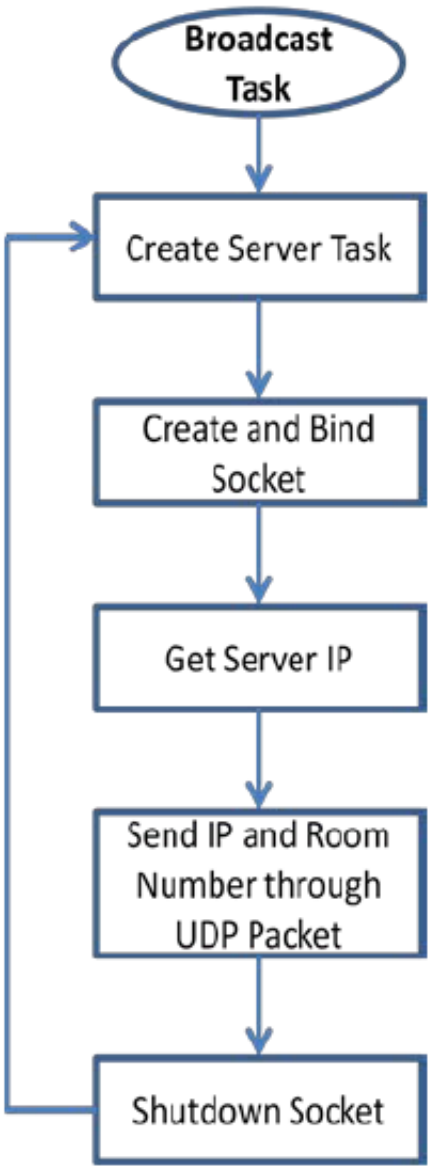

Figure 11. Broadcast task flowchart

4.3 Server task

Server task is intended to receive the information of the selected room and notify the eGUI Task that data is ready to be displayed. The server task flow can be explained as follows.

1. First, the server task creates and binds a UPD socket, then it creates the eGUI task (LCD task) and enters an endless loop.
2. In this loop, the MCU will keep listening to the UDP messages using the function `recvfrom()`. The instruction `RTCS_selectset()` is used to indicate the port that that needs to be listened.
3. When a message is received, a flag is set to indicate that data is available. This is shown in Figure 12.

NOTE

For details about the `RTCS_selectset()` function, see MQX RTCS User's Guide located in the documentation folder in MQX installation path.

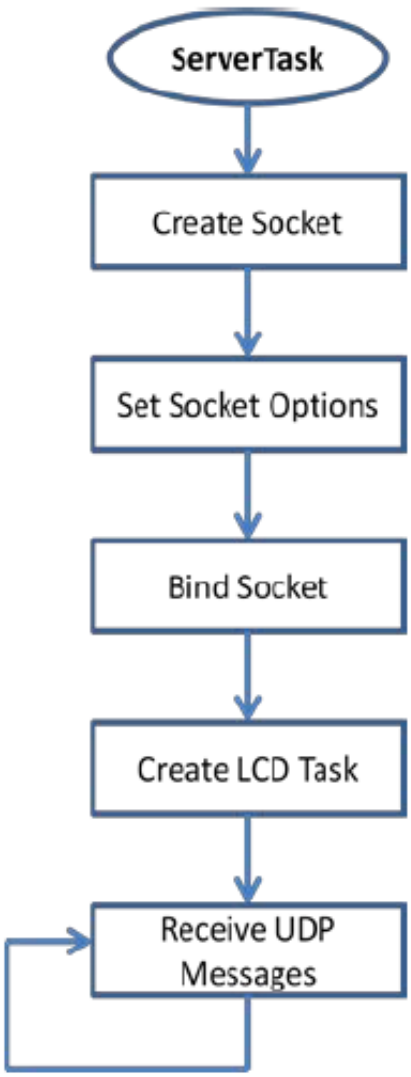

Figure 12. Server task flowchart

4.4 LCD task

The functions of the LCD task can be summarized in the following steps.

1. Installs the MQX touch screen driver
2. Creates the Time task
3. Initializes the first Screen

4. Sets the landscape orientation and enters into an infinite loop by calling the `D4D_Poll` function every 10 milliseconds as shown in Figure 13.

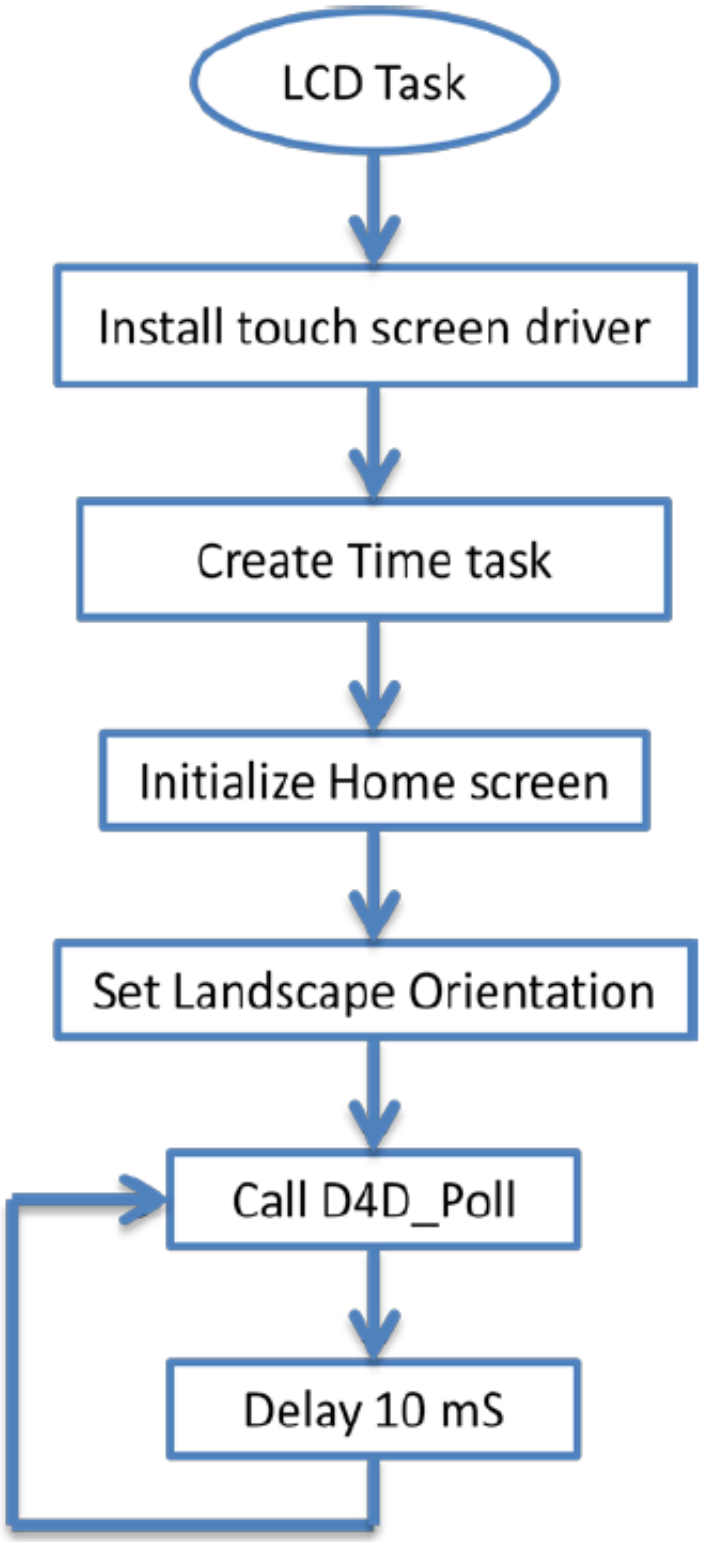

Figure 13. LCD task flow chart.

4.5 Time task

Finally, to have eGUI library working correctly with touch screens, it is necessary to periodically call the functions `D4D_TimeTickPut()` and `D4D_CheckTouchScreen()`. The purpose of this task is to enter into an infinite loop and call these functions every 25 milliseconds as shown in Figure 14. This time delay (25 ms) is enough to let the eGUI react to events from the touch screen and update the internal time flags in the eGUI library.

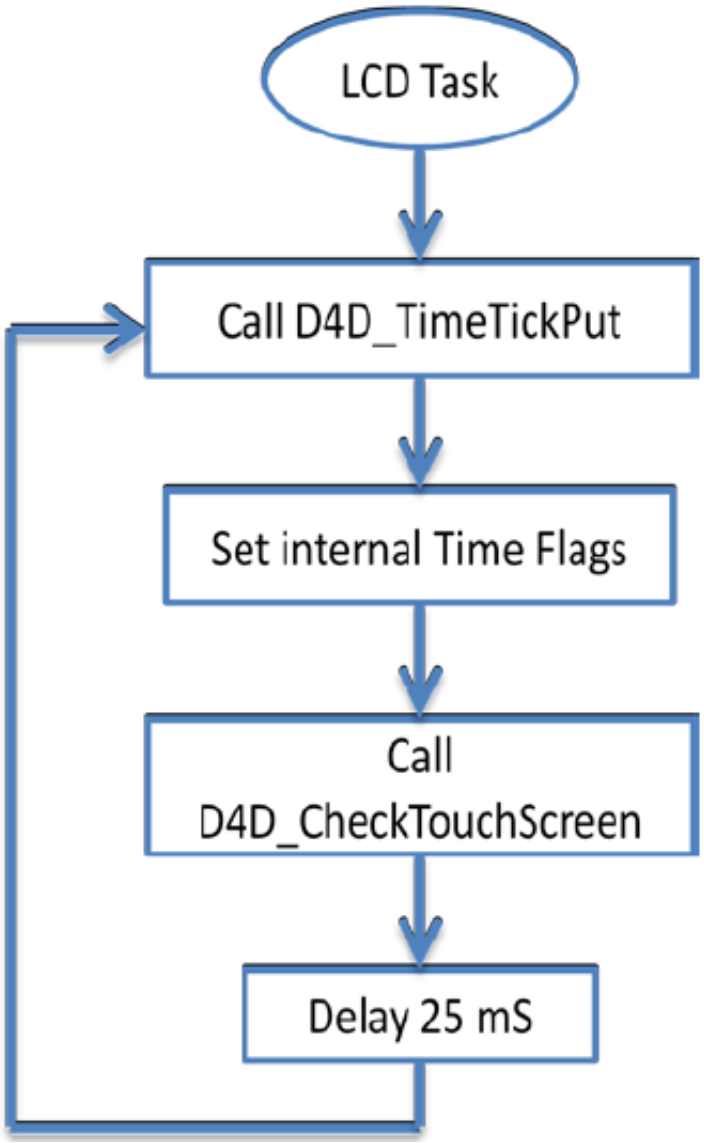

Figure 14. Time task flowchart

5 Application execution

This section provides the required steps and hardware configurations for running the application.

5.1 Required hardware

Following is the list of the required hardware:

- One Server kit
- 1-4 clients kits
- Ethernet and USB cables (one couple per kit)
- One Ethernet Switch or Router

The Sever is assembled using the following boards:

- TWR-K60N512
- TWR-LCD
- TWR-SER
- TWR-ELEV

Additionally, each client is assembled using the following boards:

- TWR-K53N512
- MED-EKG
- TWR-SER
- TWR-ELEV

For more information regarding these hardware modules, see Freescale webpage, **freescale.com**. Following is the list of the direct links for Product Summary Pages of the Tower modules used in this application note:

- K60: **freescale.com/TWR-K60**
- TWR-K53: **freescale.com/TWR-K53**
- TWR-ELEV: **freescale.com/TWR-ELEV**
- TWR-LCD: **freescale.com/TWR-LCD**
- TWR-SER: **freescale.com/TWR-SER**
- MED-EKG: **freescale.com/MED-EKG**

5.2 Server jumpers settings

The following jumper settings are required for the Server of this application.

- TWR-K60N512: Jumper J6 on position 2-3 (processor clock is taken from the TWR-SER board)
- TWR-SER: Jumper J2 (CLK_SEL) on position 3-4
- TWR-SER: Jumper J3 (CLKIN_SEL) on position 2-3 (processor clock is taken from PHY)
- TWR-SER : Jumper J12 (ETH_CONFIG) on position 9-10 to select RMII communication mode

NOTE

Both processor and serial board (TWR-SER) have to be plugged in the Tower. Processor is using external clock from Ethernet PHY on the serial card.

- TWR-LCD board: SW5-DIP[4:1] must be set to ON (enable touch screen)
- TWR-LCD board: SW1 for 16-bit FlexBus (10111110)
- TWR-K60N512: Jumper J3 on position 13-14 must be open.

5.3 Client jumpers settings

The following jumper settings are required for the Clients of this application.

- TWR- K53N512: Jumper J1 is removed to disconnect potentiometer from ADC1_DM1 input, because it is used by MED-EKG

- TWR- K53N512: Jumper 11 on position 2-3 (processor clock taken from the TWR-SER board
- TWR-SER : Jumper J2 (CLK_SEL) on position 3-4
- TWR-SER: Jumper J3 (CLKIN_SEL) on position 2-3 (processor clock is taken from PHY)
- TWR-SER: Jumper J12 (ETH_CONFIG) on position 9-10 to select RMII communication mode

NOTE

Both processor and serial board (TWR-SER) have to be plugged in the Tower.
Processor is using external clock from Ethernet PHY on the serial card.

- MED-EKG board: All jumpers in default position (using on-chip amplifiers and instrumentation amplifier external gain configured at 10x)

5.4 Running the application

This section lists the steps required for running the application.

1. First of all, it is required to have each client/server kit assembled properly.
2. It is required to have an Ethernet router with enough ports for connecting the server and all the implemented clients (up to 4). On this application, a router is required because the applications ask for a DHCP server. Ethernet hubs and switches don't perform this task.
3. Connect the Server and each Client to the router, as is shown in Figure 1 at the beginning of this document.
4. Power all the boards using mini-USB cables. Power can be supplied from TWR-ELEV, TWR-SER, or from CPU modules (OSJTAG connector). It is recommended powering the Server from TWR-ELEV, and powering the Clients from OSJTAG connector.
5. Turn on the Server and wait for the calibration screen, which will be displayed after RTCS is configured and the communication starts.
6. After calibrating the touch screen, the application jumps to the "Home" screen, from which the user can select which of the four rooms will be displayed. The "Room n" screen shows a graph of the EKG signal, and the measured Heart Rate in real time.
7. If the selected Client is not connected or powered, the application will show the message "No Data Available".
8. To obtain the heart signal, the user can either use the on-board electrodes of MED-EKG module, or connect to the external electrodes. See References section for more details.
9. To test the application running without the need of having "connected people", the user can uncomment the macro `#define TEST_EKG`. This line is included in the EKG_MQX.h file of Client's project.
10. On the same file, the macro `#define CLIENT_ID (1)` is included. It is required to change it from 1 to 4 each time a Client will be programmed. In this way, each Client will have a different identifier, allowing the application to work as expected.

6 Future work

This section provides a list of possible areas of improvement for this application.

6.1 Wireless technology

Freescale provides 3 different Wi-Fi modules for the Tower System. These modules are distributed by partners such as Atheros, GainSpan and RedPine.

freescale.com/TWR-WIFI-AR4100

freescale.com/TWR-WIFI-G1011MI

freescale.com/TWR-WIFI-RS2101

MQX provides WiFi support for TWR-M52259 and TWR-K60. As the Server in this application is developed in TWR-K60, migration from wired to wireless communication is possible.

Migrating to Wi-Fi technology with clients is not that easy. Clients are developed in TWR-K53, as Freescale does not provide TWR-K53 BSP for Wi-Fi modules; it requires creating a new BSP for this board.

6.2 Using TWR-K70

Migrating to TWR-K70 can improve the server performance as K70 provides cache memories which will decrease inherent FLASH wait states while CPU works at 100 MHz and FLASH at 25 MHz. Code will run faster and graphic algorithms will be shown faster.

Besides, the K70 can improve the performance of this application because it includes a Graphic LCD controller for using RGB screens without requiring external LCD controller, such as TWR-LCD-RGB.

6.3 Increasing the number of rooms

Increasing the number of rooms in the application involves two parts:

- Increase the number of clients IDs in EKG_MQX.h.
- Redesign eGUI application to consider the amount of rooms that the application requires.

7 References

The following references can be found at Freescale website at **freescale.com**

- MDAPPUSGDRM118 : Medical Applications User Guide, for more information about Remote Monitoring Systems
- AN4323: Freescale Solutions for Electrocardiograph and Heart Rate Monitor Applications
- AN4223: Connecting Low-Cost External Electrodes to MED-EKG, for further details on getting EKG signal by using the integrated electrodes of MED-EKG board

For additional information on Tower modules, see the following documents at Freescale website, **freescale.com**.

- TWR-K60N512-UM: TWR-K60N512 User's Manual
- TWRK60QSG: TWR-K60N512-KIT Quick Start Guide
- TWR-K53N512-UM: TWR-K53N512 User's Manual
- K53N512QSG: TWR-K53N512-KIT Quick Start Guide
- TWRLCDUM: TWR-LCD User Manual
- TWRLCDLAB: TWR-LCD Lab tutorials
- TWRLCDQSG: TWR-LCD Quick Start Guide
- TWRSERUM: Tower System Serial Module User Manual
- MED-EKGUG: MED-EKG User Manual

8 Conclusions

This application note can be used as a reference to develop Remote Monitoring applications. Besides, it provides an example of UDP communication using MQX's RTCS libraries.

It also provides useful tips to integrate eGUI into an MQX project in order to get a working application based on MQX RTOS.

Appendix A Graphic Application using eGUI

A.1 TWR-LCD communication mode

The LCD utilizes a 240 RGB x 320 QVGA display controller. The display controller is accessible to the on-board MCF51JM MCU through SPI. The controller is also accessible to any compatible Tower MCU module through SPI or the External Bus Interface (EBI) via the primary Tower Side Expansion Ports.

A.2 Freescale embedded graphical user interface (eGUI)

The eGUI/D4D is capable of generating the user menu, graphics, pictures, text, and display them on the LCD module. It allows interacting with all objects, dynamically changing, adding, or removing them. It can also read and write their status or current value. The D4D also fully supports touch screen capabilities of the LCD displays.

The eGUI stack has been specifically written with the constraints of an MCU (low FLASH and RAM) and the assumption that the graphics display RAM is write-only, as is the case of many “Smart LCD” panels. As a result, the eGUI stack can produce a stunning layered graphics display using only limited RAM and FLASH from the MCU and a small library footprint.

A.2.1 Structure of project with eGUI

The following figure shows the position of the D4D in the whole project. It is placed between low-level drivers of the LCD and the user application.

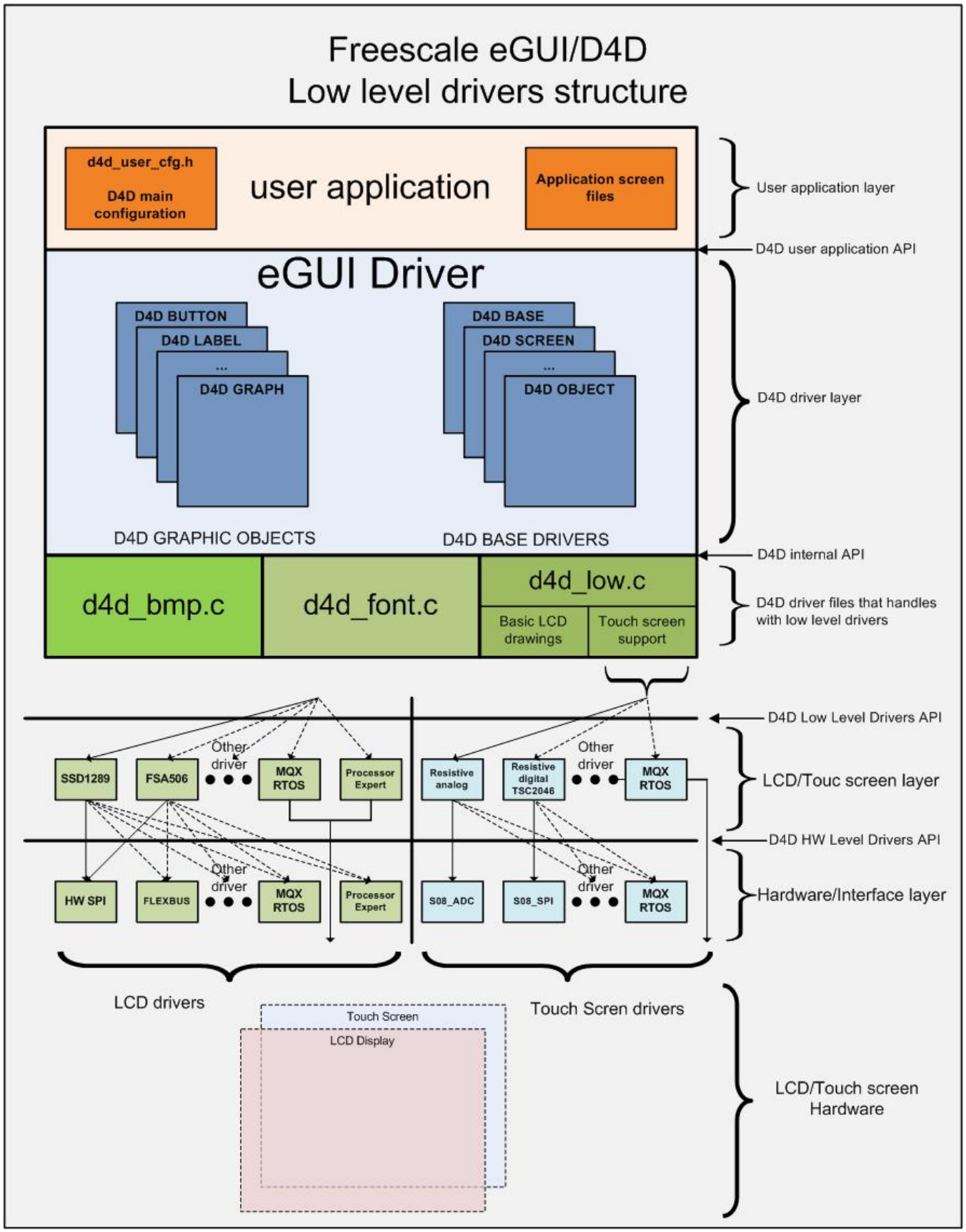

Figure A-1. Freescale eGUI/D4D block diagram

A.2.2 File structure

The following figure shows the file structure of eGUI/D4D, created from five types of files:

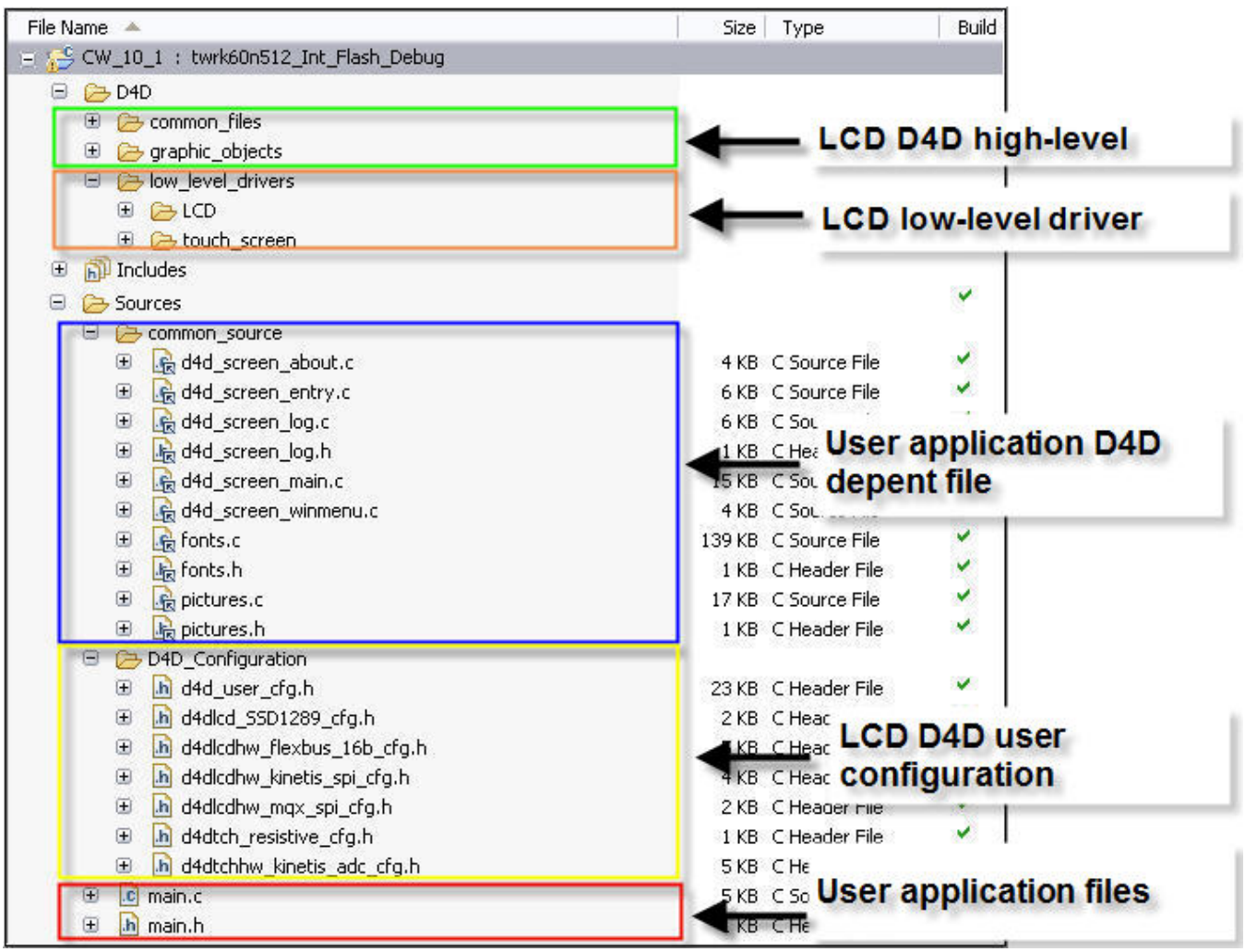

Figure A-2. eGUI/D4D file structure in CodeWarrior project.

The above project can be downloaded from Freescale Embedded GUI Software, available on **freescale.com.**

A.2.3 Adding eGUI library to a MQX RTCS application

Using the Freescale Embedded GUI Software, the user must add LCD D4D high-level driver, LCD low-level driver for Kinetis, and LCD D4D user configuration to the MQX RTCS application. The user can add D4D files either using “add files…” by drag-and-drop or right-clicking to the destination folder.

NOTE

These steps also apply to any MQX Project using CodeWarrior 10.2.

1. Add LCD D4D high-level driver to the RTCS MQX project. Once the files from Freescale_embedded_GUI_SW.zip file are extracted, the LCD D4D high-level driver files (common_files and graphic_objects folders) can be found at the D4D folder.

2. Since the Kinetis K60 device and TWR-LCD module are used for this project, the user needs to add only the low-level drivers for Kinetis K60 device and the screen module of the TWR-LCD. The following figure shows the drivers that must be added.

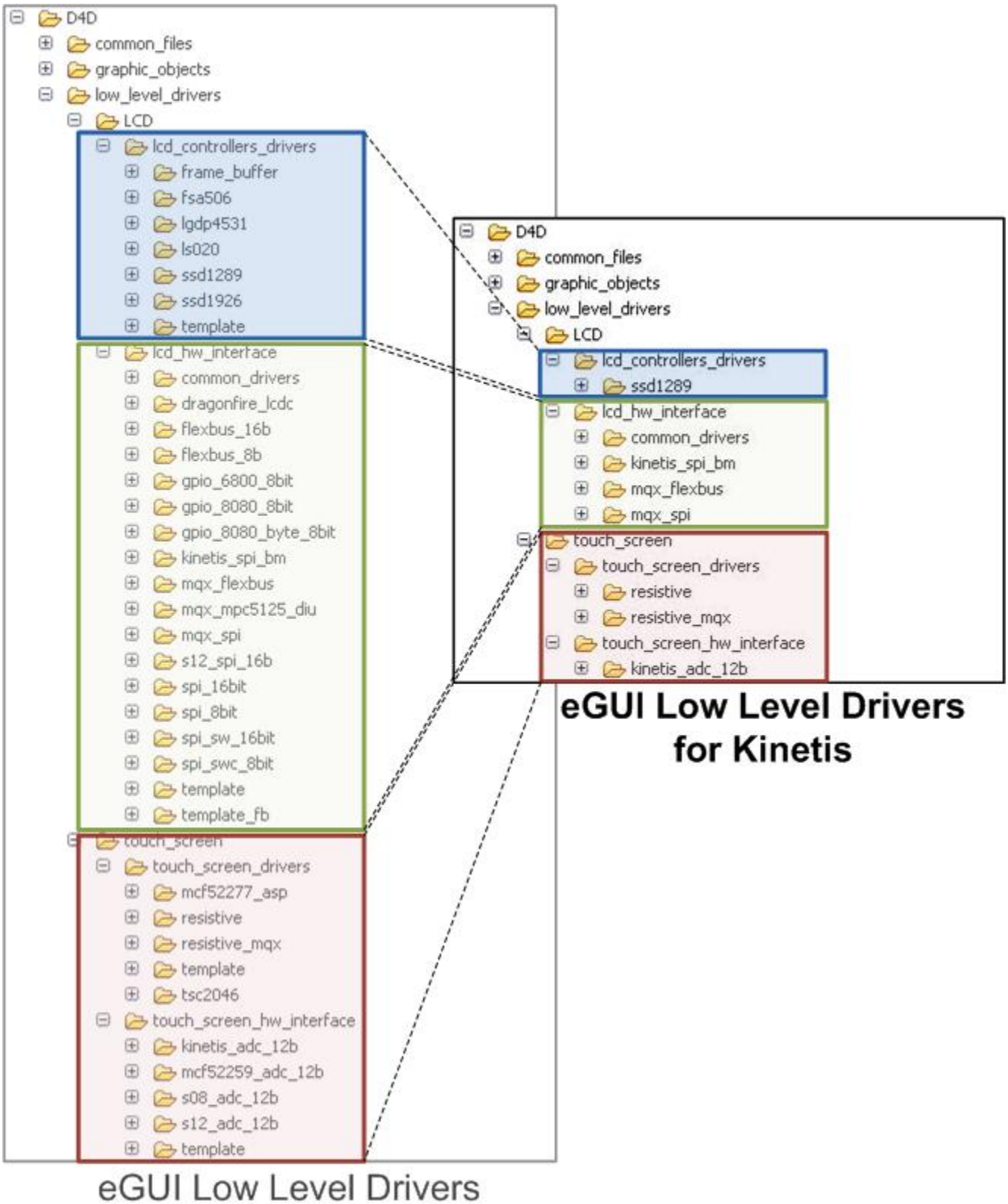

Figure A-3. eGUI low-level drivers for Kinetis.

3. Add LCD D4D user configuration files to the Sources folder of the MQX RTCS project. The LCD D4D user configuration files (D4D_configuration) can be found at this path …*_Official_Demos\EGUI_D4D_Demo \TWR_K60N512\MQX_3_7\CW_10_1\Sources.*
4. Include the D4D library into the project by adding the below paths in the compiler settings (right-click and select Properties of the MQX RTCS Project). Figure A-4 shows the windows settings where the path must be added.

```
"${PROJECT_LOC}/D4D"
"${ProjDirPath}/Sources"
"${PROJECT_LOC}/Sources/common_source"
"${PROJECT_LOC}/Sources/D4D_Configuration"
```

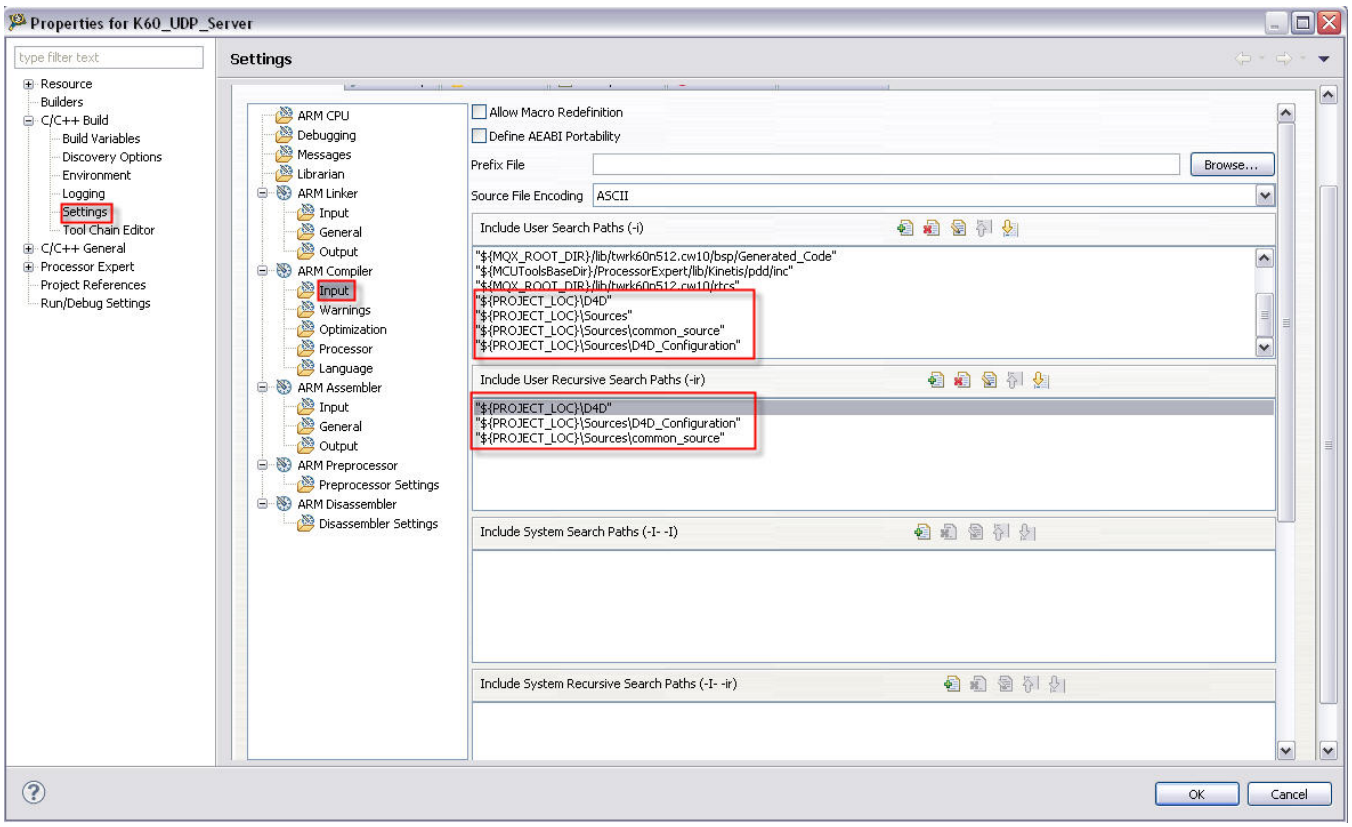

Figure A-4. User and recursive paths

5. Add the eGUI tasks to the MQX RTCS project. There are two tasks that must be added to keep eGUI alive:
   - Time task: This task periodically checks for events every 25 ms.
   - LCD task: This task handles the D4D display. The MQX RTCS project was made on MQX 3.8. The MQX developer team changed the touch screen driver (TCHRES) and added the feature to use two different ADC modules for reading X/Y axis. For this reason, the LCD task must be modified to work on MQX 3.8
6. Create the first Screen. The basic item of D4D structure is SCREEN. The screen represents the real one screen shown on LCD.
   The screen contains:
   - List of all used objects on screen.
   - Functions bodies:
     - OnInit – One time called function with first use of screen
     - OnActivate – Function called on each activation of screen
     - OnDeactivate – Function called before deactivation of screen
     - OnMain – Function is called periodically when screen is active
     - OnObjectMsg - Function is called with all system messages of this screen

Create a new folder named as “common_source” in Source folder as shown in the figure below.

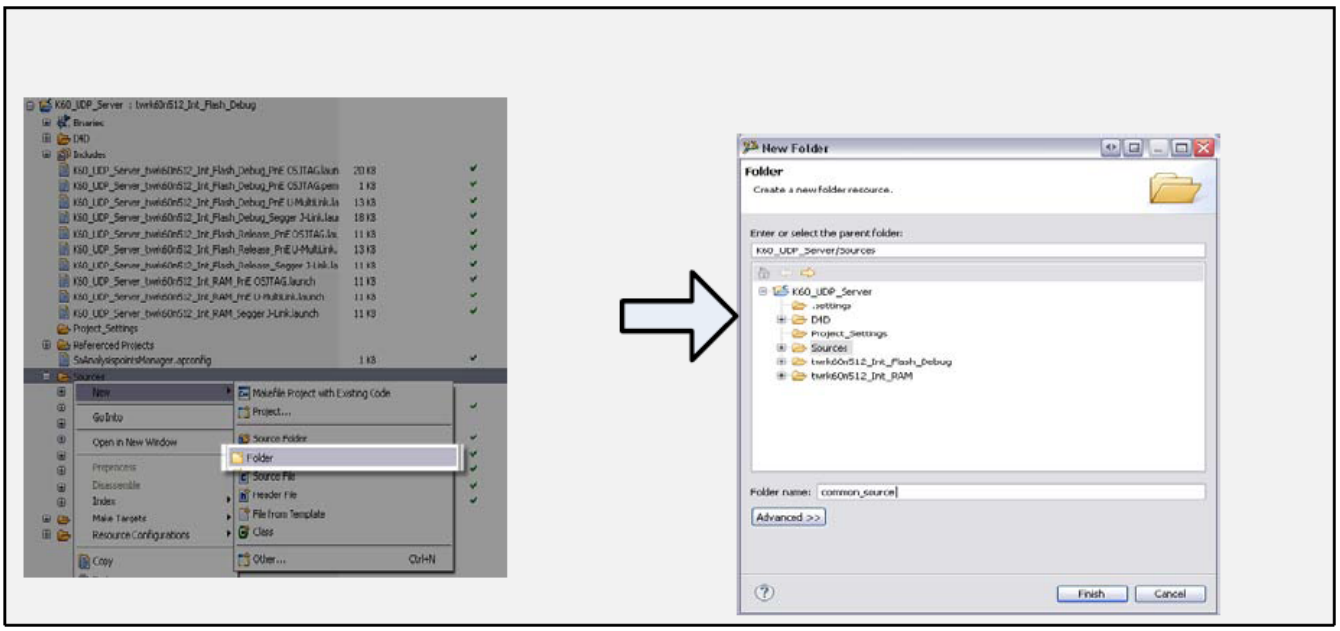

Figure A-5. Create a new folder.

1. Create a new c file named as “HOME.c” in common_source folder as shown in the following figure.

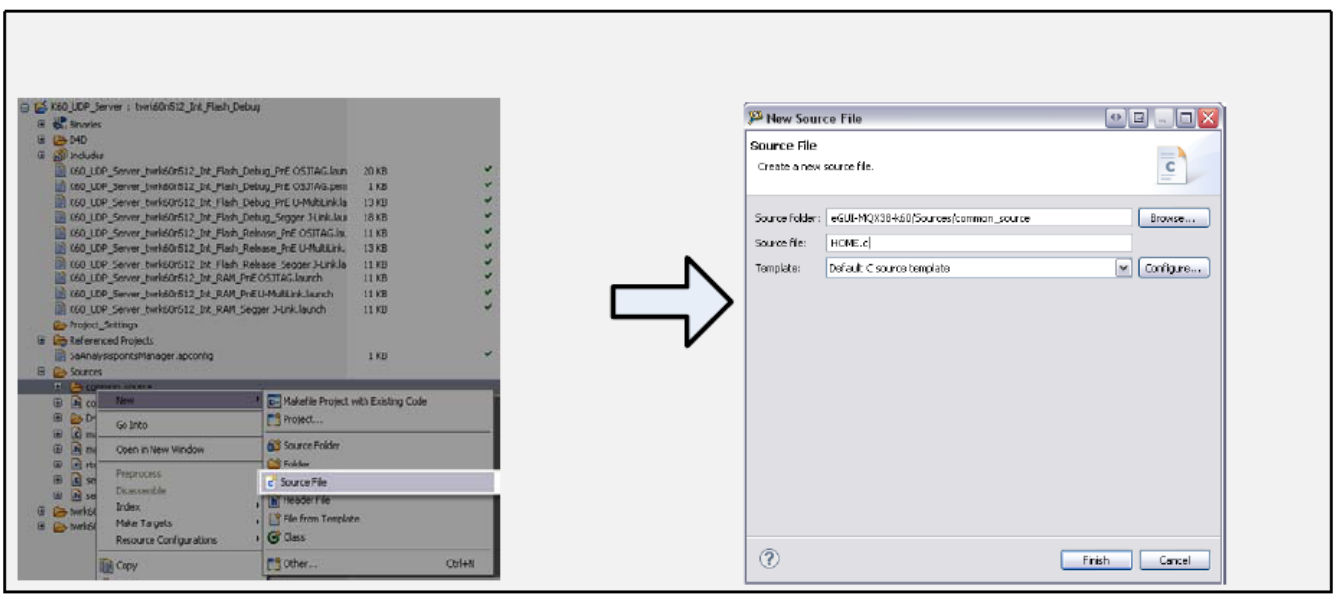

Figure A-6. Create a new C file.

2. Open the d4d_screen_template.c.template file at this path …*\D4D\configuration_example* from Freescale Embedded GUI software, downloaded from **freescale.com.**
3. Copy the code from d4d_screen_template.c.template file and paste it into HOME.c file.
4. Replace the name of the template screen from screen_template to screen_home.

```
D4D_DECLARE_STD_SCREEN_BEGIN(screen_home, ScreenHome_)
```

5. Find and replace of all the functions bodies from ScreenTemplate_ to ScreenHome_ .
6. Add font.c and font.h from this path *_Official_Demos\EGUI_D4D_Demo\common_source* to common_source folder. font.c and font.h files contain two font types: Arial and Berlin Sans. For this project, only Arial and Berlin Sans fonts are used but using the Freescale Embedded GUI Image Converter Utility, the user can add installed Windows fonts. This utility can be downloaded from **freescale.com** searching for Freescale Embedded GUI Image Converter Utility.

7. At this point, the project structure must be like as shown in the following figure.

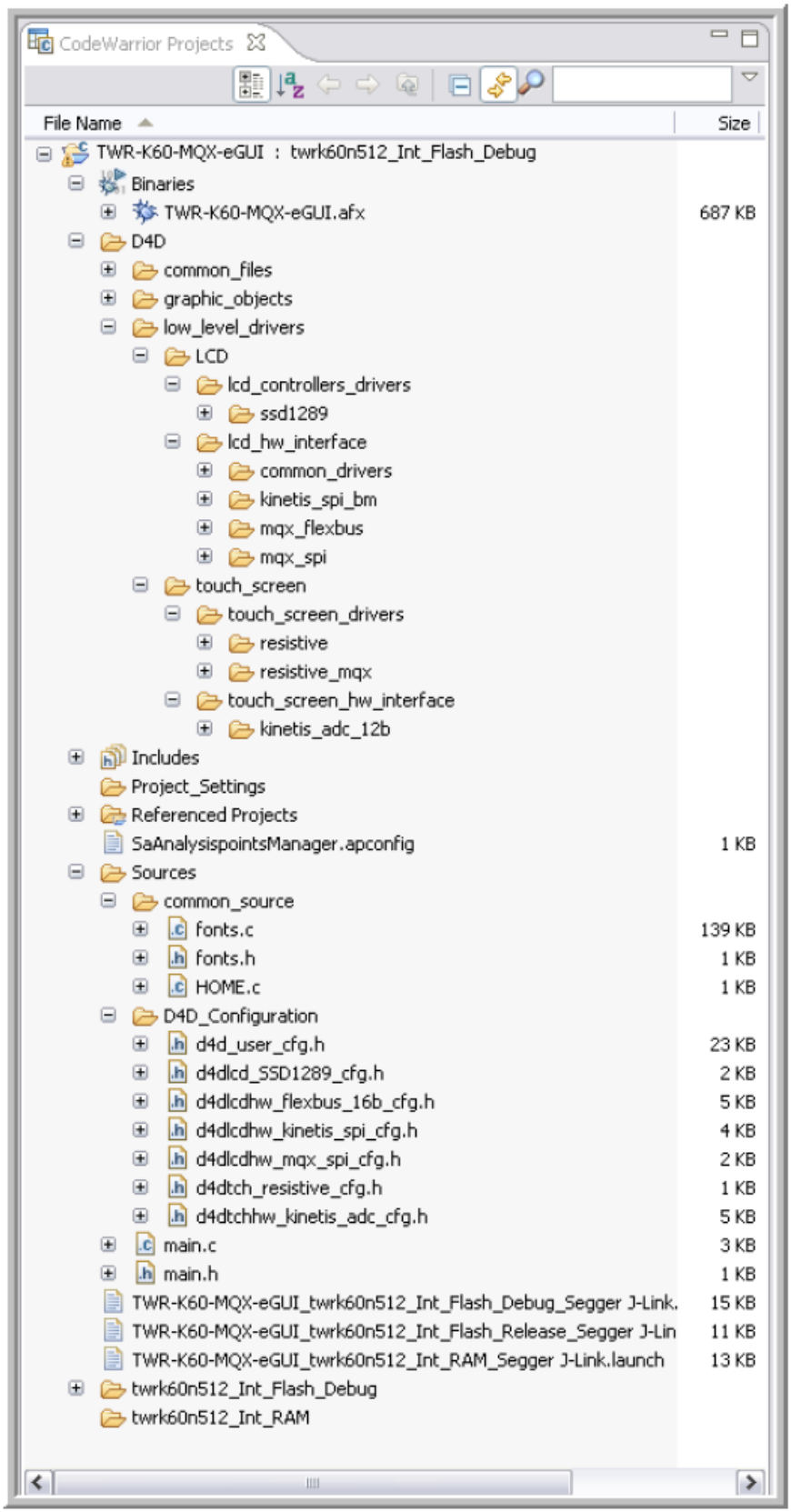

Figure A-7. MQX project with eGUI library.

8. Build and debug MQX project as shown in the figure below.

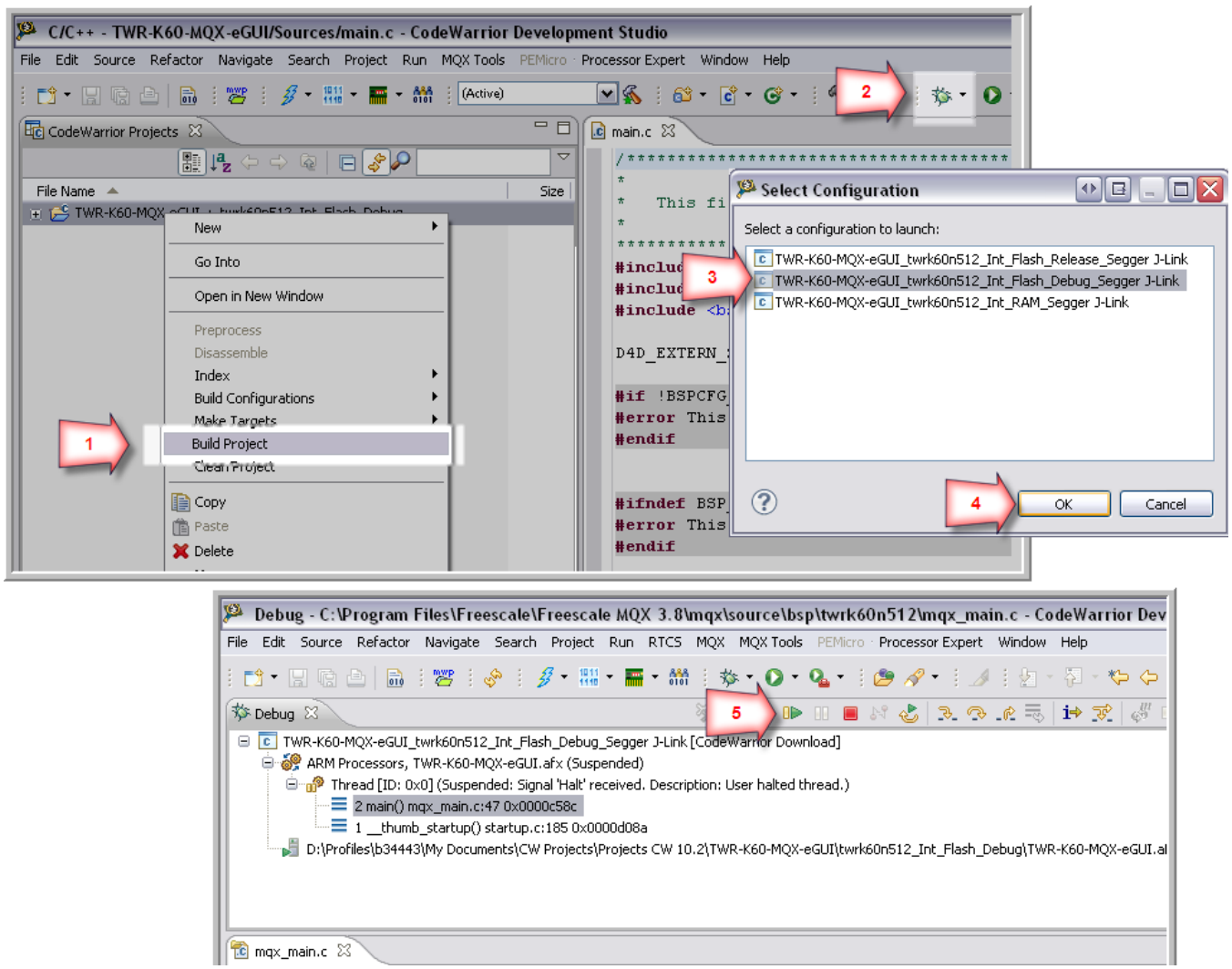

Figure A-8. Debugging a project.

9. When the user clicks the Resume icon, the TWR-LCD will display the calibration screen as shown in the following figure.

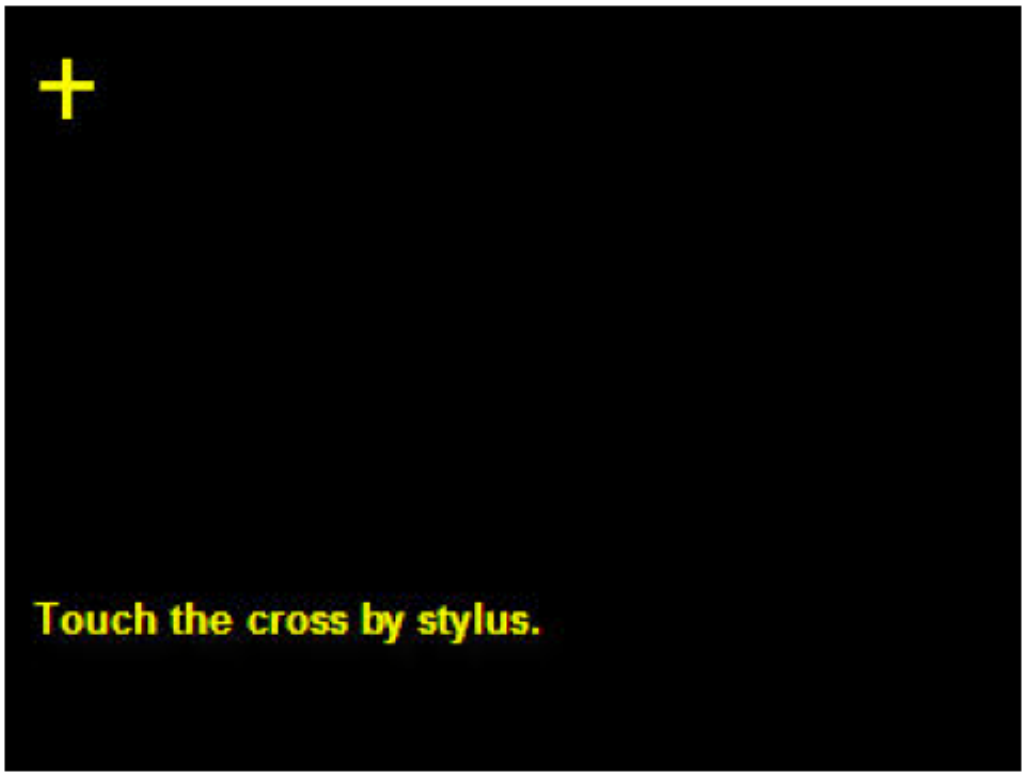

Figure A-9. eGUI screen calibration.

A.2.4 Developing a graphic user interface for the MQX RTCS application using eGUI library

At this point, it is possible to add as many screens and D4D objects as the user application requires.

The following section explains how to build five screens and how to use the objects needed in this particular application.

A.2.4.1 Creating the Home screen

This section explains how to build the home screen which is the screen that is shown on the TWR-LCD after the RTCS is initialized and the server binds a socket.

Using D4D_Button object, five buttons are added into the Home screen. These buttons are used to navigate among the five screens (1 button for Home screen and 4 for each of the rooms). D4D_PICTURE is used to add a picture as the home screen background. The following figure shows the Home screen.

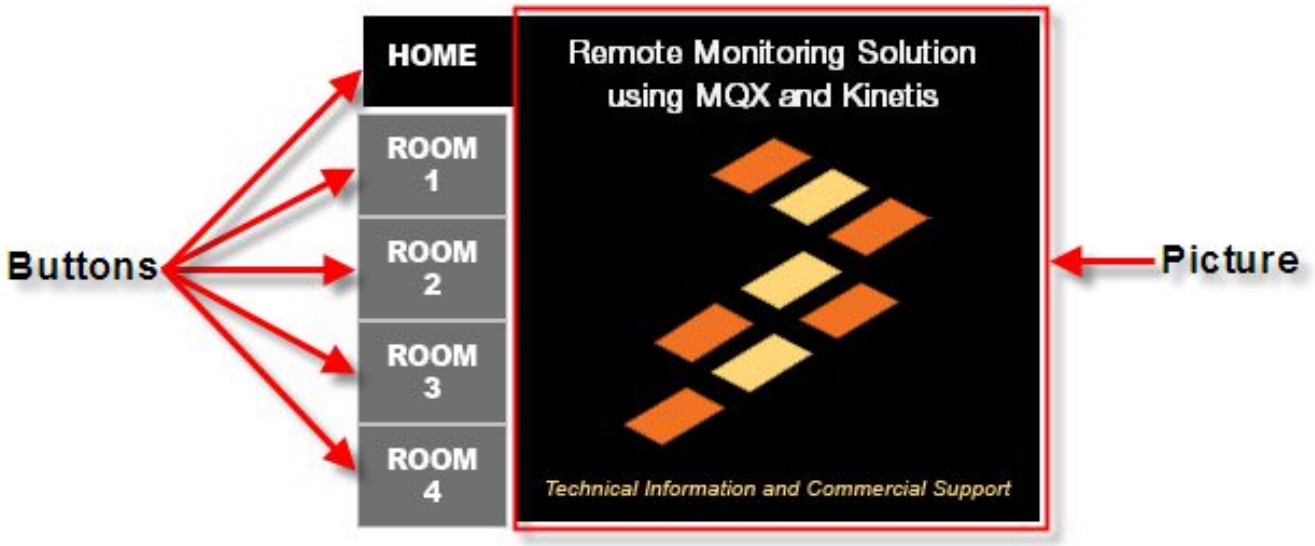

Figure A-10. Home screen.

1. D4D_PICTURE is an object used to show a picture on the screen. The Home screen background is shown in Figure A-11. This picture was created using a graphic editor software and was converted using Freescale Embedded GUI Image Converter Utility, that can be downloaded from **freescale.com.**

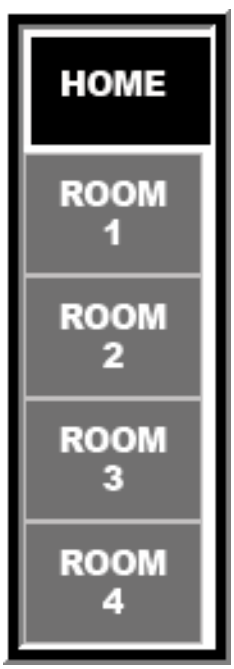

Figure A-11. Home screen background picture.

To add a picture, the macro D4D_DECLARE_STD_PICTURE is defined. As an example, to add the Home screen background picture, the macro is defined as follows.

```
D4D_DECLARE_STD_PICTURE(HOME,73,0,&bmp_Screen_Home)
```

- The name of the picture is HOME.
- The position is 73 in X-axis and 0 in Y-axis.
- The pointer to the bitmap array of the picture is bmp_Screen_Home.

NOTE

To convert an image into an array; Freescale provides a tool called Embedded GUI Image Converter Utility which can be downloaded from **freescale.com**. Also, see EGUICUG : Freescale Embedded GUI Converter Utility 2.0 Quick - User's Guide, available on **freescale.com.**

2. D4D_BUTTON object is intended to be used as a standard button and on this particular project is used to switch among different screens. Each of the five buttons of the Home screen has its own background picture, and as the Home Screen background, the buttons' background pictures were converted using "Freescale Embedded GUI Converter Utility". The button background pictures are shown in the following figure.

Figure A-12. Button background pictures in Home screen.

To add buttons, the macro D4D_DECLARE_STD_BUTTON_AUTOSIZE is used. The Room 1 button is described as an example below:

```
D4D_DECLARE_STD_BUTTON_AUTOSIZE(Room1_SH,NULL,
0,48,&bmp_Room1_Button_Inactive,&bmp_Room1_Button_Inactive,NULL,Room1_Button)
```

- The name of the button is Room1_SH.
- There is no text used on this button.
- The position is 0 in X-axis; and 48 in Y-axis.
- The pointer to the bitmap array of the button is bmp_Room1_Button_Inactive.
- There is no font used on this button.
- The callback function when this button is pressed is Room1_Button.

The button object contains a few predefined constants that are used in a standard button declaration.

The screen behavior and visual aspect flags:

- D4D_BTN_F_DEFAULT: This is a help macro that is used for default configuration. The value of this macro is defined as follows:

  ```
  D4D_OBJECT_F_VISIBLE |
  D4D_OBJECT_F_ENABLE |
  D4D_OBJECT_F_TABSTOP |
  D4D_OBJECT_F_TOUCHENABLE |
  D4D_OBJECT_F_FOCUSRECT
  ```

  For this particular application, the D4D_BTN_F_DEFAULT flags were changed as below:

- D4D_BTN_F_DEFAULT value:

```
D4D_OBJECT_F_VISIBLE |
D4D_OBJECT_F_ENABLED |
D4D_OBJECT_F_TABSTOP |
D4D_OBJECT_F_TOUCHENABLE |
D4D_OBJECT_F_FASTTOUCH
```

D4D_BTN_F_DEFAULT flag can be modified in d4d_button.h file.

NOTE

For more information about the D4D flags, see eGUI/D4D Public Predefined Init General Object Flags section from the Freescale Embedded GUI User Manual that can be downloaded from **freescale.com** searching for "Freescale Embedded GUI (D4D)".

A.2.4.2 Creating the Room screens

This section explains how to build the Room screen. Only Room 3 screen is explained as rest of the room screens are duplicated. The difference among the Room screens is that Room 1 shows data from client ID 1, Room 2 shows data from client ID 2, and so on. When Room 3 is active, it sends a signal to client ID 3 for starting the transmission and the rest of the clients are in stand-by.

As in Home screen, the Room screens have a five-button column to navigate the screens.

- D4D_Picture: This object is used to show the Room screen background.
- D4D_Label: This object shows the beats per minute and it is updated every 100 milliseconds.
- D4D_Graphic: This object prints the QRS complex sent by the client ID 3.

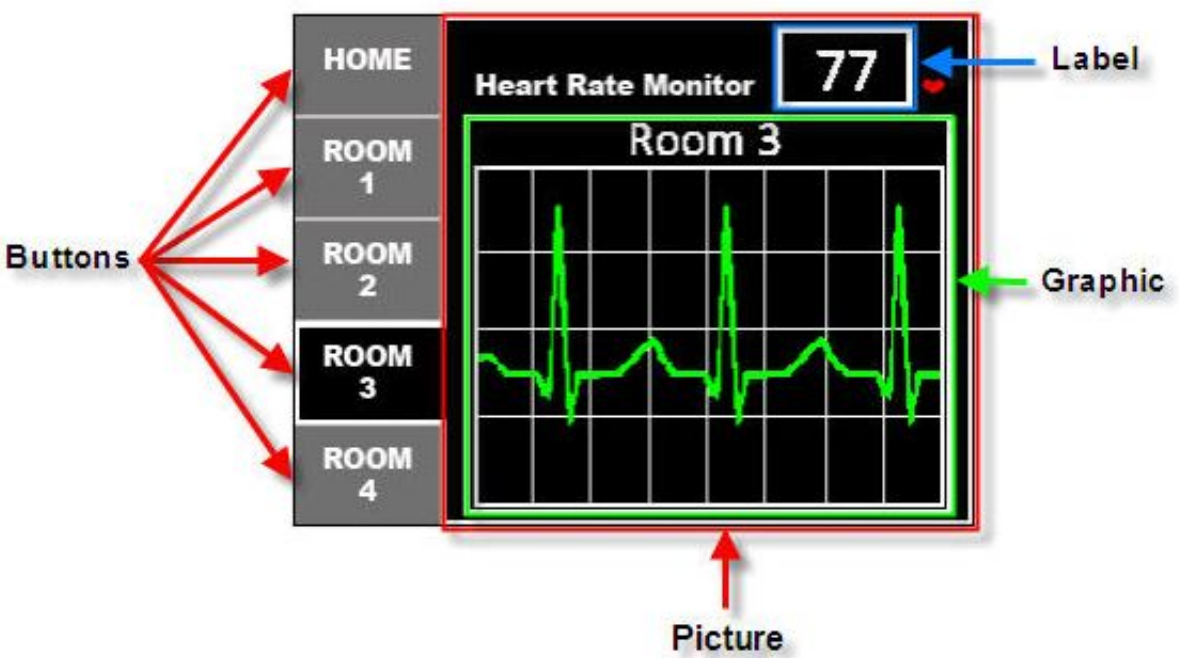

Figure A-13. Room screen

1. D4D_Picture: The Room screen background is shown in Figure A-14 and the D4D_Picture configuration is the same as in home screen background in Creating the Home screen.

Figure A-14. Room screen background

2. D4D_Button: The button's background pictures are shown in Figure A-14 and the D4D_Button configuration is explained in Creating the Home screen.
3. D4D_Label: This object is prepared to be used as a visualization object to show simple text one-line information of an application in a graphical form. In this particular application, label shows the heart beats rate and is updated every 100 milliseconds.
   To add label object, the macro D4D_DECLARE_LABEL is used. The label object is described below:

```
D4D_DECLARE_LABEL(BPM_R1, " ", 230, 5, 62, 40, D4D_LBL_F_DEFAULT, NULL,
FONT_BERLIN_SANS_FBDEMI12_BIG, NULL, NULL)
```

   - The name of the label is BPM_R1.
   - The string is " " on this label.
   - The position is 230 in X-axis; and 5 in Y-axis.
   - The size of the label is 62x40 pixels.
   - The flags have the value of the macro D4D_LBL_F_DEFAULT.
   - The default color scheme is used.
   - The font type for the label is Berlin Sans 12.
   - There is no callback function.

   Every 10 milliseconds, the `Room1_OnMain()` function is called and updates the label value received from the client ID 1 with D4D_SetText as below:

```
D4D_SetText(&BPM_R1, bpm);
```

   Where &BPM_R1 is the pointer to the label object and bpm is the string with the value received from the client ID 1.

4. D4D_Graph object is prepared to create a simple graph. The graph object definition macro is created from three individual parts that allow indicating a complete graph object with various counts of graph traces:
   - Begin part—This part specifies all the necessary parameters of the graph object itself.
   - Add trace to graph—This part allows to use multiple D4D_DECLARE_GRAPH_TRACE macros to add all traces into the graph.
   - End part—This part is used only to close the graph traces array definition.

   In this project, graph object shows the QRS complex signal in one trace. This signal is sent by the client ID 3 every 10 milliseconds.

```
D4D_DECLARE_STD_GRAPH_BEGIN(room1_graph, "Room 1", 85, 50, 220, 185, 8, 4, 20,
FONT_ARIAL7_WIDE, FONT_7)
D4D_DECLARE_GRAPH_TRACE(dataTraceR1, D4D_COLOR_GREEN, D4D_LINE_THICK,
D4D_GRAPH_TRACE_TYPE_LINE)
D4D_DECLARE_GRAPH_END()
```

- The name of the graph is room1_graph.
- The string that appears as graph title is “Room 1".
- The position is 85 in x-axis; and 50 in Y-axis.
- Size of the graph is 220x185 pixels.
- The number of grid lines is 8 in X-axis and 4 in Y-axis.
- The Length of data buffers is 20.
- The font type for the graph title is FONT_ARIAL7_WIDE.
- The flags have the value of the macro D4D_GRAPH_F_DEFAULT
- The name of the trace is dataTraceR1.
- The color of the trace is green.
- The type of the trace D4D_LINE_THICK.

For this particular application, the D4D_GRAPH_F_DEFAULT flags were changed as below:

```
D4D_GRAPH_F_DEFAULT value:
D4D_OBJECT_F_VISIBLE |
D4D_OBJECT_F_ENABLED |
D4D_OBJECT_F_FOCUSRECT |
D4D_GRAPH_F_MODE_ROLLOVER
```

D4D_GRAPH_F_DEFAULT flag can be modified in d4d_user_cfg.h file.

The rest of the Room screens are duplicated with Room1 screen.

For the complete application project, see AN4644SW.zip file, available on **freescale.com**.

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: AN4644
Rev. 0, 01/2013