Home Automation System using LPC1769

Daniel Schermuk  <daniel.schermuk@gmail.com>
Ezequiel Espósito  <ezequiel.esposito@gmail.com>
Federico Roasio  <federoasio@gmail.com>

First Semester - 2011
Abstract

This project aims to create a multipurpose standalone device capable of controlling peripherals, such as lights, fans, sockets, etc. and monitoring sensors (temperature, humidity, etc). The controller can be configured both by an onboard touch screen or an Ethernet connection, showing the active devices and previous settings. The target of this device is firstly domotics, but can be used in telemetry, feedback controls, etc.
Contents

1 Introduction 2

2 Description of the device 3
  2.1 Block Diagram and Network Hierarchy ......................... 3
  2.2 Implemented Functionalities .................................. 4

3 Hardware description 5
  3.1 Central Unit circuit ........................................... 5
    3.1.1 Microprocessor ........................................... 5
    3.1.2 Touch-GLCD ................................................. 5
    3.1.3 Ethernet interface ................................ ........ 6
    3.1.4 Serial Port ................................................ 6
  3.2 Dimming module circuit ........................................ 6
    3.2.1 Analog Front-End .......................................... 7
    3.2.2 Analog Back-End .......................................... 7
    3.2.3 Serial port ................................................ 8
    3.2.4 Microprocessor ............................................ 8

4 Firmware description 9
  4.1 Central unit module ............................................ 9
    4.1.1 Application specific source code .......................... 10
    4.1.2 RTOS library ............................................. 12
    4.1.3 Ethernet library .......................................... 12
    4.1.4 Domotic library ........................................... 14
    4.1.5 GLCD library ............................................. 14
    4.1.6 CMSIS and MCU libraries .................................. 16
  4.2 Light dimming module .......................................... 16

5 Conclusion 18
1 Introduction

Home Automation (also known as Domotics) is the automation of the home. This kind of systems may control lighting, house appliances, air conditioning, irrigation and other systems, in order to improve comfort, energy efficiency and security.

This kind of systems is particularly useful for the disabled or elderly, improving the life quality and avoiding special aid expenses.

The Home Automation System integrates electrical devices in the house with each other in order to control domestic activities such as pet feeding, smart lighting or entertainment systems. This devices are often connected through a computer network, allowing them to be controlled by any personal computer or mobile device (smartphones, tablets, etc.). The communication technologies used in the systems are mostly determined by the house topology. Pre-existing houses use PLC (powerline communication) or wireless networks (radio or IR), while new homes can be outfitted for dedicated wiring through the walls, avoiding some interference issues and lowering the final cost.

Home automation is a wide spread idea, leading to the well expected “Intelligent Home” from science fiction movies and novels. Sadly, the elevated costs, incompatible standards and complexity, resulted in a small market share, limiting these systems to hobbyists or the wealthy.
2 Description of the device

2.1 Block Diagram and Network Hierarchy

The Home Automation System network developed in this project consists in two elements:

- Central Controller Unit: this device is the brain of the network. The user can control or monitor all the peripherals through the onboard graphical screen or by the embedded web server.

- Lighting Peripherals: this device operates the lights of the house, according to the settings made in the controller.

Each element’s block diagram is shown in the next figures:

![Central Controller Unit block diagram](image1.png)

Figure 1: Central Controller Unit block diagram.

![Lighting peripherals module block diagram](image2.png)

Figure 2: Lighting peripherals module block diagram.
The network topology is shown in the Figure 3. As already mentioned, the implemented elements are the central unit and the lights dimmer peripheral.

![Network topology diagram]

Figure 3: Network topology.

### 2.2 Implemented Functionalities

The developed project implements the following functionalities:

- **Central Unit** with onboard touch graphical screen, with an easy and user friendly GUI.
- **Embedded web server** for remote monitoring and controlling peripherals.
- **Peripheral unit** for light bulbs (and any generic resistive load such as a heater), with dimming capabilities.
3 Hardware description

3.1 Central Unit circuit

The circuit used for the Central Unit is showed in the next figure.

![Central Unit circuit](image)

Figure 4: Central Unit circuit.

3.1.1 Microprocessor

The hardware presented in this section uses LPCXpresso1768 from Embedded Artists. This board comprises a target board with an LPC1768 microcontroller, combined with a JTAG debugger for easy debugging of the system. The following MCU features will be used:

- Analog/Digital Converter to digitize the analog signal coming from the touchscreen.
- Digital IO to control the GLCD.
- Ethernet interface.
- UART interface to communicate with the light dimming module.

3.1.2 Touch-GLCD

The touch-GLCD block is composed of a touchscreen and a GLCD. The GLCD is based on the KS0108 controller. It is driven using 13 lines, 8 for data and 5 for control. Port PIO0 of the LPC will be used for this purpose. Since the output voltage level from LPC is 3.3V and the GLCD has 5V inputs, it is necessary to use a buffer to adjust levels between these two. The interface is unidirectional, so the buffer was easily implemented using ULN2803 inverters.
A 4-wire resistive touchscreen was used. It is connected in differential mode, so 2 ADC (ADC2 and ADC3 from LPC) were needed to read the Y and X coordinates respectively. While reading the Y coordinate, the two leads from the X coordinate operate in digital mode, causing a voltage difference across the screen that is read by ADC2. The same happens while reading the X coordinate.

3.1.3 Ethernet interface

In order to provide the Ethernet interface it’s mandatory to have two main layers:

- Logical Layer: this layer implements the logical hardware needed for the protocol to work.
- Physical Layer: this layer implements the hardware for the physical connections. It modulates the logical signals to be sent through the wires.

The Logical Layer relies on the LPC1769’s on chip Ethernet MAC, which has the necessary hardware to handle ethernet connections. The Physical Layer used in this project utilizes the 10/100M Ethernet PHY (LAN8720) found in the LPCXpresso stick, only needing a RJ45 jack with magnetics.

3.1.4 Serial Port

The serial port interface is an UART interface. It consists of 2 standard lines Tx and Rx for data transmission and 1 line for GND connection. We will use a wired interface, but we expect to replace it with a Xbee module in the near future.

3.2 Dimming module circuit

The circuit used for the light dimming is showed in figure 5. It consists of an analog front-end, a microprocessor, an analog back-end, and a serial port.

![Figure 5: Dimming module circuit.](image-url)
3.2.1 Analog Front-End

The analog front-end generates a signal that is synchronized with the zero point of the AC power supply. In this case a 220 VAC and 50Hz power line was used. However, the hardware (and the software also) is prepared to work with other power lines, for example 120 VAC and 60Hz.

The heart of the front-end circuit is the H11AA1 optocoupler. It consist of two gallium-arsenide infrared emitting diodes connected in inverse parallel driving a single silicon phototransistor output. A 100kohm resistor is used to limit the current in both diodes. Whenever the current across the diodes is greater than 100mA, the diodes emit light and turn on the output phototransistor, which is connected as an inverter. When the input AC signal is near zero volts both diodes are turned off, so the inverter’s output is high.

![H11AA1 schematic](image)

Figure 6: H11AA1 schematic.

3.2.2 Analog Back-End

The analog back-end isolates the microprocessor and the triacs which drives the load. It consists of a random-phase optoisolator triac driver MOC3021 followed by a BTA08 triac, suitable for general purpose AC switching.

![MOC3021 schematic](image)

Figure 7: MOC3021 schematic.

The light intensity is controlled varying the AC conduction angle. Activating the triac at different phase angles will allow us to control the portion of the total AC sine wave applied to the load and, thereby, regulate the power flow to it. The next figure illustrates the concept of the phase control.
In this circuit the “hot” side of the line is switched and the load connected to the ground side. The 39 ohm resistor and 0.01 $\mu F$ capacitor are for snubbing of the triac, and the 470 ohm resistor and 0.05 $\mu F$ capacitor are for snubbing the coupler. These components may or may not be necessary depending upon the particular load used, resistive or inductive.

### 3.2.3 Serial port

The serial port interface is an UART interface. It consists of 2 standard lines Tx and Rx for data transmission and 1 line for GND connection.

### 3.2.4 Microprocessor

The MCU used for this module is again a LPCXpresso1769 from Embedded Artist. The following MCU features will be used:

- Rising edge external interrupt to synchronize with the AC zero crossing points.
- Timers to generate the delay phase angles.
- Digital outputs to fire the triacs.
- UART interface to communicate with the Central Unit.
4 Firmware description

The firmware of the system is divided mainly in two modules:

- The central unit module
- The peripheral module

In this application note a peripheral module capable of controlling different sources of light was implemented. This is just an example of a possible application of the system. In the next paragraphs the peripheral module will be referred as "light dimming module".

The main project consists of two source subprojects (one for the central unit module and one for the light dimming module).

4.1 Central unit module

The central unit’s project module comprises the application level the static libraries level.

The application level includes:

- The application specific source code
- The RTOS library
- The Ethernet library

The static libraries level includes:

- The Domotic static library
- The GUI static library
- The Touch Panel static library
- The CMSIS and MCU static libraries

A block diagram of the central unit module’s structure is showed in figure 9. The figure details the relationship between the most important blocks of the central unit module.
The project’s files hierarchy (code organization) is showed in figure 10.

In next subsections the source code of the central unit module is detailed

4.1.1 Application specific source code

The application source code is placed in “.../Application/App”. The code included in that location uses the static libraries to generate the specific application behavior.

The startup logic of the central unit module is defined in “.../Application/main.c”.

Figure 9: Block diagram of the central unit module

Figure 10: Files Hierarchy
Firstly the Domotic’s objects and the GUI’s objects are created, and the respective libraries are started. Next the resources of the RTOS and the different tasks are created. After that, the Touch Panel static library is configured and initialized. The central unit module must be capable of communicating with the peripheral devices. For this reason the UART module of the LPC1769 is configured. The Ethernet module doesn’t need to be initialized because there is a specific RTOS task created in charged of this. Once that all the modules are properly initialized and the scheduler of the RTOS is started and the application begins to run.

In "/Application/App/Domotic" the Domotic static library is used. The Domotic’s objects for the specific application are created here, two dimming lights and two on/off lights were instantiated in this case. Two functions were implemented for this purpose: StartDomoticLib() and CreateDomoticObjects(). The first one is in charged of starting the Domotic static library indicating what functions are responsible of allocating and destroying dynamic memory.

The second one uses the Domotic static library functions to create the specific objects needed for the application. In the current library version only “on/off” and “dimming light” objects can be created. It’s also defined the handlers that catch the events triggered by the domotic objects.

In "/Application/App/GUI" the GUI static library is used. The GUI’s objects for the application are created here. The objects are displayed inside a window in the graphic LCD when the user of the GUI static library call them. Two functions were implemented for the purpose of initializing the library and creating the objects StartGUI() and CreateGUI(). The first one is in charged of starting the GUI static library. It defines what functions are responsible of allocating and destroying dynamic memory. The second one uses the GUI static library functions to create the specific objects needed for the application. Several windows with different buttons, texts and sliders were created. The main idea is to generate a graphic user interface that allows the user to interact with the Domotic objects.

In "/Application/App/Touch" the Touch Panel static library is used. In this section the handler of the external interrupt EINT3_IRQHandler() and the RTOS’s task Task_TouchPanel() were defined. The external interrupt triggers an event when the touch panel is activated. The event is handled by EINT3_IRQHandler(). This function gives an RTOS semaphore for unblocking the task Task_TouchPanel(). In this task the function TouchPanel_Read() is used. This function reads the touchscreen and decides which event is produced by the user (click, double-click, drag-drop, etc.). When the decision is taken the Task_TouchPanel() function calls the GUI_Window.RaiseEvent() function (that belongs to the GUI static library) with the information of the event that was produced over the screen. If the GUI library decides that the event belongs to an object, it triggers an event to the application in order to execute the code generated by the user of the GUI static library. Finally the Task_TouchPanel() take the semaphore and goes to the blocked state. The process starts all over again when the semaphore is given by the external interrupt.

In "/Application/App/uIP" the RTOS’s task vuIP_Task() is declared as external. The topics related with this task are explained later in this application note.

In "/Application/App/RTOS" we can find the semaphore that synchronizes the handler of the external interrupt, called EINT3_IRQHandler(), with the RTOS’s task Task_TouchPanel(). This synchronization is used to avoid having a periodic task running to detect activity on the touch panel.
The functions used to communicate with the peripherals are defined in "../Application/App/Devices". In this specific application the UART module is used to make the communication between the main central unit and the peripheral module.

In "../Application/App/Bitmaps" the bitmap of the presentation of the faculty is defined. This bitmap is used by the application in the start up window showed in the graphic LCD.

4.1.2 RTOS library

The Central Unit firmware runs a real time operative system called FreeRTOS\(^1\). This RTOS was ported to run on LPC1769. The OS runs in preemptive mode, allowing multiple tasks to run in parallel. In order to handle asynchronic data input, both the touchscreen and the ethernet port work with interrupts. These interrupts only store the input data in buffers, and wait for the dedicated task to process it. The unit has two tasks running on the OS:

- ulP\(_\text{task}\)

- TouchPanel\(_\text{Task}\)

\textbf{ulP\(_\text{task}\)}: This task runs periodically processing input data previously stored in specific buffers. It synchronizes the events related with the ethernet communication.

\textbf{TouchPanel\(_\text{Task}\)}: This task rises the corresponding events (click, double click, etc.) to be handled by the graphic library. The task is aperiodic.

4.1.3 Ethernet library

\textbf{General Description} The Central Unit implements an embedded webserver which allows to remotely change settings and monitor activity. This webserver is based on the one used in the FreeRTOS Demo\(^2\), using the ulP TCP/IP Stack\(^3\) and the Protosockets Library\(^4\) in order to allow linear coding and making code more readable. This libraries provide all the basic protocol to handle connections and retrieve the requested data. A layer description is provided in the next figure:

\(^1\)http://www.freertos.org/
\(^2\)http://www.freertos.org/LPC1768\_RedSuite.html
\(^3\)http://www.sics.se/ adam/old-uip/
\(^4\)http://www.sics.se/ adam/old-uip/protosockets.html
In order to describe the data flow, we must analyze synchronous and asynchronous events separately:

- Asynchronous Events: When any Ethernet connection is detected, an interrupt handler takes care of it, storing the request data in a Buffer.

- Synchronous Events: the uIP Task is periodically executed, processing the Buffer’s data and rising the corresponding events.

Since peripheral information is dynamically generated, we provided the HTML_rep library, which implements two main functionalities:

- Dynamic Code Generation: the function generate_html_rep generates the HTML code needed to represent a given peripheral, considering its state and properties.

- Input Data Processing: the process_html_input function looks for input data and rises an event which is then handled by its specific handler.

**Objects Representation** In order to provide a user friendly interface, we developed a standard representation for each peripheral type. In this project we only used dimming capable devices and standard ON/OFF switching devices. Their representation is shown in the next figures.
Each representation gives a unique name for each button, making possible further input data processing. This name is based in every peripheral’s ID, which is based in it’s position in the Objects List.

**Input Processing** All the data flow from the web browser to the central unit is sent in HTML’s *get* input type. This data management sends input variables and text fields content as a web page request. This incoming request is then stored in the uIP buffer and waits the uIP_Task to process it. When the uIP_Task gets executed, it looks for data in the Buffer and sends it to a dedicated function charged with it’s processing, called *process_html_input*. This function searches for button and textfield names and the IDs associated to them, gathers the new information and rises events to the corresponding peripherals.

### 4.1.4 Domotic library

The *Domotic Library* implements two main structures for the project:

- **Peripheral Description Objects**: These objects model each peripheral type, according to it’s properties.

- **Active Peripheral List**: This list includes every installed peripheral. Each node has an instance of a Peripheral Description Object.

**Peripheral Description Objects** As already told, these objects model the peripherals. Each peripheral implements a constructor and have the necessary attributes to correctly represent the peripheral. In order to provide software layers isolation, the library implements setters for each variable attribute.

**Active Peripheral List** This list consists in enlaced nodes, holding the information for the installed devices. Each node can be created and added to the list with the implemented functions. The list also implements functions for searching nodes according to their name, give an ID to the node according to its position in the list or search for the node for a given ID.

### 4.1.5 GLCD library

The GLCD (graphic LCD) library is composed of four static libraries:

**Lib_GLCDLowLevel**: The communication with the GLCD hardware is solved in this library. An interface to separate the library for a specific microcontroller is implemented here. The video memory (a matrix proportional to the number of pixels of the GLCD) is also allocated at this level. Basically, every time that something must be drawn in the GLCD the video memory is modified and a specific function of the GLCD is called in order to send the video memory to the GLCD.
Lib_GLCDPrimitives: In this static library the graphic primitives are solved (dot, vertical line, horizontal line, diagonal line, square, etc.). All the functions coded in this static library change the video memory that is placed inside the Lib_GLCDLowLevel. The video memory is sent to GLCD hardware whenever is desired.

Lib_GLCDgui: The GUI (Graphical User Interface) is implemented in this library. It implements the following objects: window, text, text slider, picture and button. In order to draw these objects in the video memory the primitives of the Lib_GLCDPrimitives library are used.

Lib_TouchPanel: The logic necessary to catch the events (click, double click, drag and drop, etc.) that are produced over the touch panel is implemented in this library. In the current version only the detection of the click event is implemented. In this library we can also find the functions that configure the hardware to take the x-y coordinate of the events. The application that uses this library must configure an external interrupt that is triggered when the touch panel is touched. The interrupt calls the function “TouchPanel_Read” until this function returns true. When the function returns true, the event is detected and the its information is sent to the Lib_GLCDgui library. The Lib_GLCDgui library searches over all the objects in order to determine if the event belongs to someone. If an object is matched, then the object’s event is triggered in the application so that the specific user code is allowed to be executed.

Figure 14: Home window
Figure 15: Credits window
Figure 16: Menu window
Figure 17: Light Dimming window
4.1.6 CMSIS and MCU libraries

This project uses the Cortex Microcontroller Software Interface Standard (CMSIS) \(^5\), that gives a hardware abstraction layer, simplifying the code. A second library called \textit{Lib\_MCU} was included. This library is based on the ported CMSIS for LPC17xx \(^6\), with only subtle changes, and gives a simpler access layer to peripherals, timers, etc. making it easier to read the code.

4.2 Light dimming module

The basic flow chart of the light dimming module firmware is showed next.

![Light dimming module firmware flow chart.](image)

The software is really simple. Every time a zero crossing is detected, one timer per channel is initialized with a match value in accordance with the desired light dimming value. When the time value is matched, the respective triac is fired. Both zero crossing detection and the timers work with interrupts.

The dimming values are updated via a serial communication implemented with the UART3 of the LPC1769. We used a simple communication protocol to transmit both the channel and its respective dimming value. Every time a character is received, an

---


\(^6\)http://ics.nxp.com/support/lpcxpresso/
interruption flag is activated and we save the character in a FIFO buffer implemented by software. After that we process the information stored in the buffer, and update the dimming value.
5 Conclusion

A home automation system was designed. This project shows how domotics appliances could be developed using a LPCXpresso Board. In this case, we focussed on the user interface. We provided not only a touchscreen with GLCD user interface, but also a web interface by means of an ethernet connection. This allows the user to interact with different home peripherals in an user friendly way.

As an example, we also implemented a light control module, with dimming capabilities. This module shows how power devices can be controlled with a LPCXpresso using optocouplers.

Finally, we could make all the modules work together through the use of a RTOS.

References


