Mission 5: Real-Time project on a "naked" computer

Tommaso Marinelli Matteo Di Pirro

December 20, 2017

1 User manual

2 Documentation for system engineers

3 Documentation for programmers

3.1 Specification

The program implements a DHCP relay. Figure 1 depicts how it works.

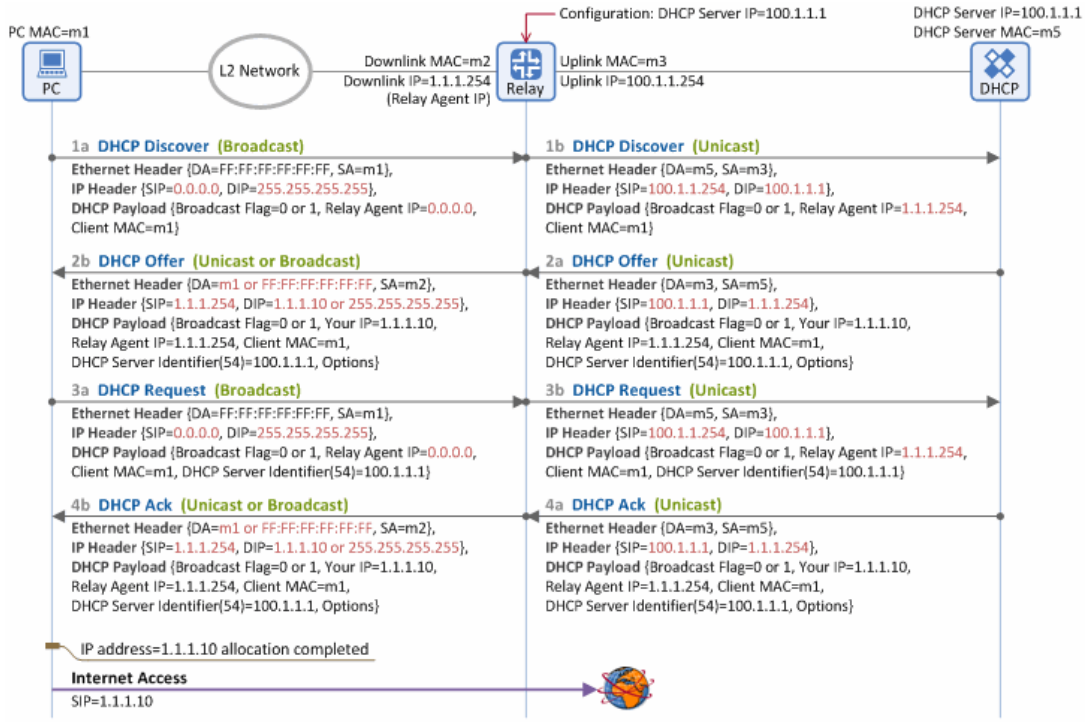


Figure 1: Example of DHCP protocol communication[1]

Basically, a DHCP relay is a "man in the middle" between the client and the DHCP server. It helps the client to contact the server in order to obtain an IP address. It does so receiving the broadcast packets sent by clients and forwarding them in an unicast connection to one (or more) DHCP servers, and vice versa, forwarding the server responses in broadcast to the clients.

While modeling its behavior, we refer to the above figure. We suppose to be able to detect when a message comes from a client and when it comes from a server by means of two different interruptions, without looking at the `OPCode` field (which is 1 for a request and 2 for a response).

[1] `https://www.netmanias.com/en/?m=view&id=techdocs&no=6000`

In other words, two *different events* occur. This is reasonable, since Figure 1 depicts two different links (*Downlink* and *Uplink*), with two different MAC and IP addresses.

Our ASG diagram is depicted in Figure 2.

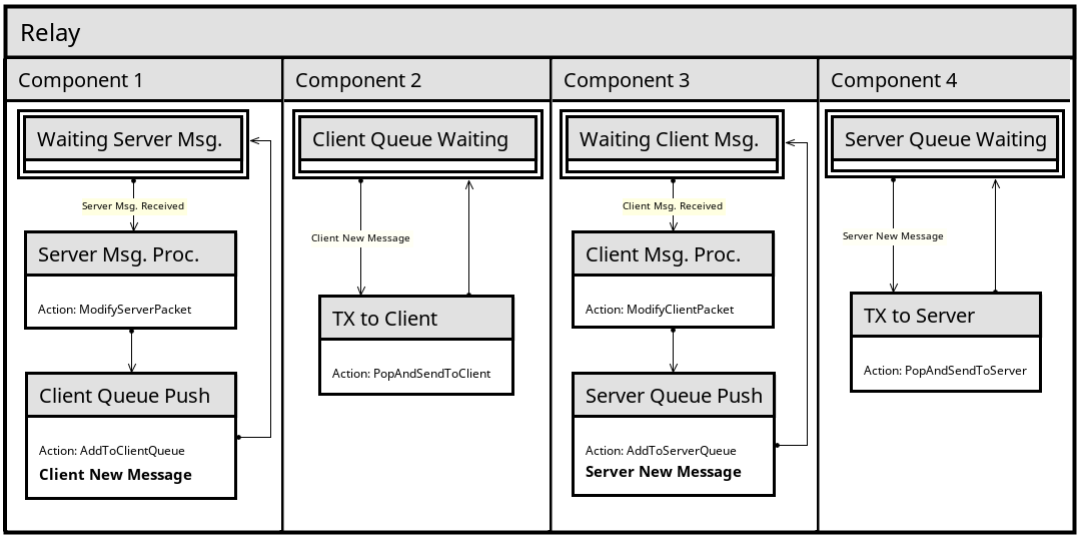


Figure 2: ASG diagram modeling a DHCP relay

The DHCP relay consists of four parallel components following a producer-consumer pattern. Two components, 1 and 3, wait for a packet, either from a client (broadcast) or a server (unicast). They then modify this packet as follows:

- **Component 1** - `ModifyServerPacket`:
  - **Ethernet Payload**
    - Destination MAC Address: DHCP Relay Uplink MAC → Broadcast
    - Source MAC Address: DHCP Server MAC → DHCP Relay MAC Address
  - **IP Payload**
    - Source IP Address: DHCP Server IP Address → DHCP Relay Downlink IP
    - Destination IP Address: DHCP Relay Downlink IP → Broadcast
- **Component 2** - `ModifyClientPacket`:
  - **Ethernet Payload**
    - Destination MAC Address: Broadcast → DHCP Server MAC
    - Source MAC Address: PC MAC Address → DHCP Relay Uplink MAC
  - **IP Payload**
    - Source IP Address: 0.0.0.0 (no IP address) → DHCP Relay Uplink IP
    - Destination IP Address: Broadcast → DHCP Server IP
  - **DHCP Payload**
    - Gateway IP Address (GIADDR): 0.0.0.0 → DHCP Relay Downlink IP

The two components then push the modified packet in a queue (two different queues are present, respectively for server and client) and set a rendez-vous (respectively **Client New Message** and **Server New Message**) to awake the corresponding consumer.

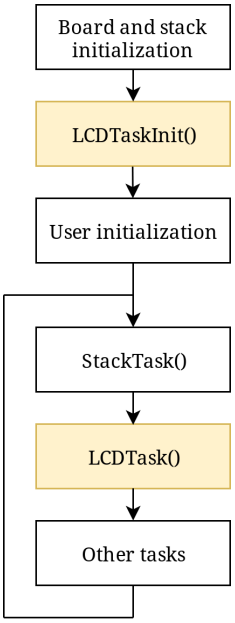
Component 2 and 4 represent the consumers. If they are waiting, they are awoke by a push of the corresponding producer. Once a message is received, they enter the transmission state, and leave it only when the queue is empty. Here they continuously pop an element from the queue and send it in broadcast (Component 2) or in unicast to one (or more) DHCP servers (Component 4).

This separation in two producers-consumers allow us to separate two logical different operations: handling the traffic incoming client-side and the one incoming server-side.

3.2 LCD non-blocking module

The key concept in multitasking real-time systems is that tasks must be "small enough", in order not to prevent the processor for executing other operations. In this project it is required to use the LCD display to print information messages, but the C functions provided by Microchip are not optimized for multitasking and take a non negligible time to be processed; the `LCDBlocking` module has been thus converted in `LCDNonBlocking` and integrated into the TCP/IP stack to provide a better display handling in an environment with different tasks.

The most time-demanding functions in the LCD library (`LCDInit`, `LCDErase`, `LCDUpdate`) have been **split in smaller states**. When called, the LCD task can only execute a part of a function: the following ones are executed at successive task calls; furthermore, each state can be accessed only if a sufficient amount of time has passed from the previous one, in line with what happened in the original file where some delays were placed between instructions.



The main functions of the new library are `LCDTaskInit` and `LCDTask`. The first one is an initialization function which is used to assign default values to control variables and configure the resources needed for the correct operation handling. The latter is the proper task, which runs inside the cooperative multitasking loop.
These functions are called in the main entity as a design choice, but they could have also been integrated into the `StackTsk` file of the TCP/IP stack.

The big difference between the new library and the old one is that the operations are not executed immediately but are “appended” somewhere, so that the task can pick and perform them in successive steps without losing information about other operation requests happening in the meanwhile.
In order to do this, a **circular list** (actually, a circular array of structures) has been implemented: each time an *Init*, *Erase* or *Update* operation is required, the list is filled with a code representing that operation and the text to write, if needed; this guarantees that the display always reflects the correct history of operations regardless of the state of the shadow copy.
The circular list static allocation required the availability of more than 256 bytes in memory, that is the maximum dimension allowed by default due to internal division of databanks in the PIC18. To overcome this limitation, the linker script was modified to create a single databank of twice the size and a `#pragma` directive was introduced to memorize the list in that exact memory location; the solution was fully tested and does not introduce any kind of issue.

The delay handling is entrusted to **Timer 1** (external 32.768 kHz oscillator) because it is not used by any other part of the system; the timer is in a bounded configuration and the initial value of the register is calculated according to the necessary minimum delay for each stage. If such a waiting condition is active, LDC task checks for an overflow of the timer register before proceeding to the execution.

The instruction flow of the LCD task is represented in the next page; it is a simple diagram which is not meant to explain the detailed content of each block but the general behavior of the module. For in-depth analysis it is possible to read the code.

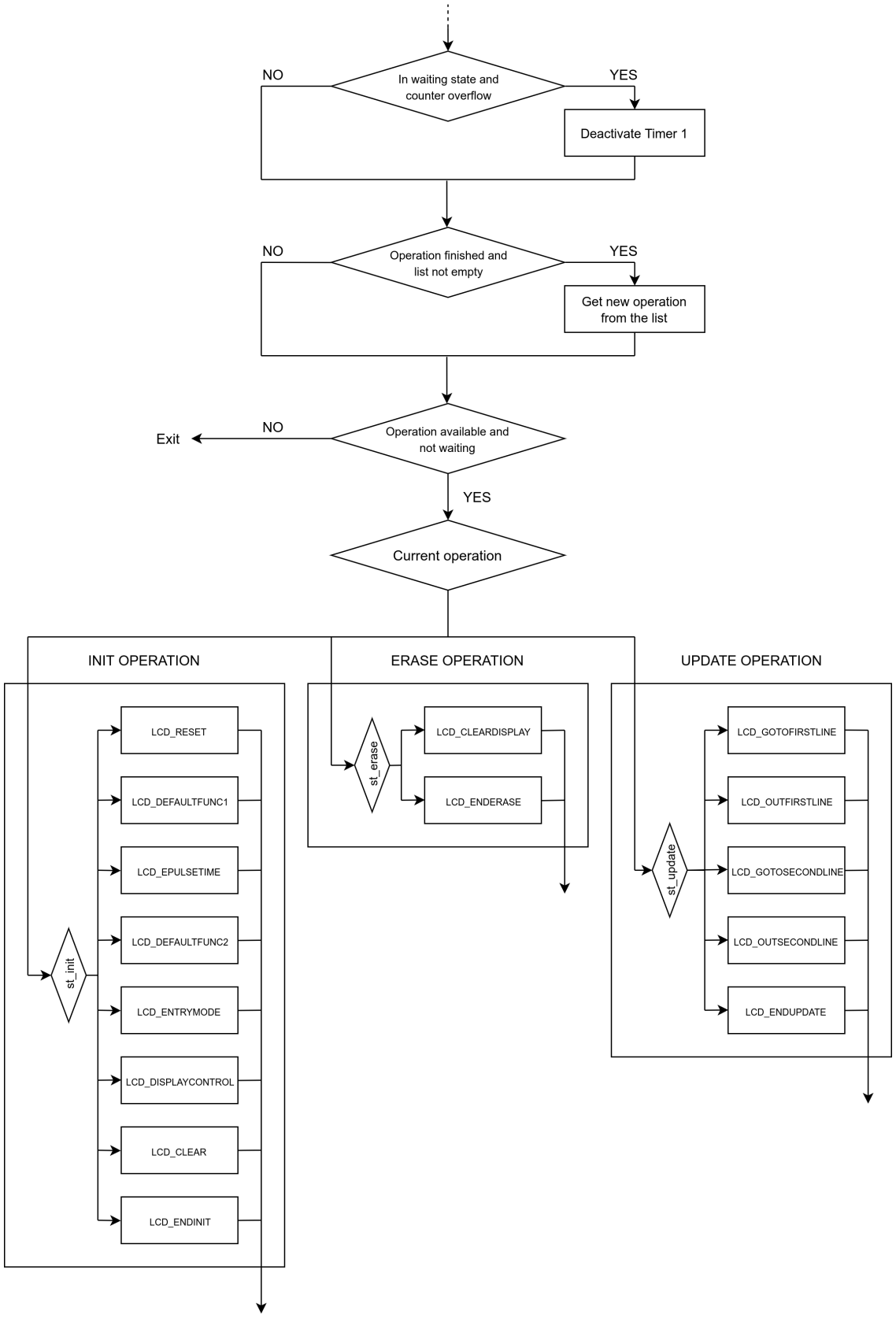


Figure 3: LCD task instruction flow chart

4 Program listings