Test Sample
Project Overview

Table of Contents

Requirements

1.) An embedded electronic device with the following peripherals:

a.) 1 spring-loaded button that may be pushed and held by a user.

b.) 1 Ethernet interface. Assume that the driver is functional, and the device has a valid IP address and is Internet-accessible. BSD sockets are available to use in the application.

c.) 1 white LED, controllable by GPIO.

d.) 1 red LED, controllable by GPIO.

e.) 1 Battery monitor, read via ADC. Units read are in units of millivolts.

f.) 1 USB battery charger interface, which may be plugged and unplugged at any time to charge or discharge the battery. Assume that all battery charging functionality is independent and properly working.

2.) When a user pushes the button, the white LED shall illuminate. The white LED shall be illuminated for as long as the button is held down. When the button is released, the white LED shall turn off.

3.) Network Application

a.) While the button is pushed, a UDP socket shall be opened with a server at the domain "test.ring.com" on port 13469. Every 1 second, the device shall send a packet of 2 bytes to the server with a counter, starting at value 0 and increasing by 1 every packet.

b.) For every 2-byte UDP packet sent to the server, the server shall return back a 2-byte packet on the same port echoing the counter. If no echo is received within 500ms or the value returned from the server is not equal to the value sent from the device, the device shall re-send the current value. This shall continue until the correct value is received from the server, at which point the device will increment the counter and proceed as normal.

c.) When the button is released, the socket shall be closed and the counter shall be reset.

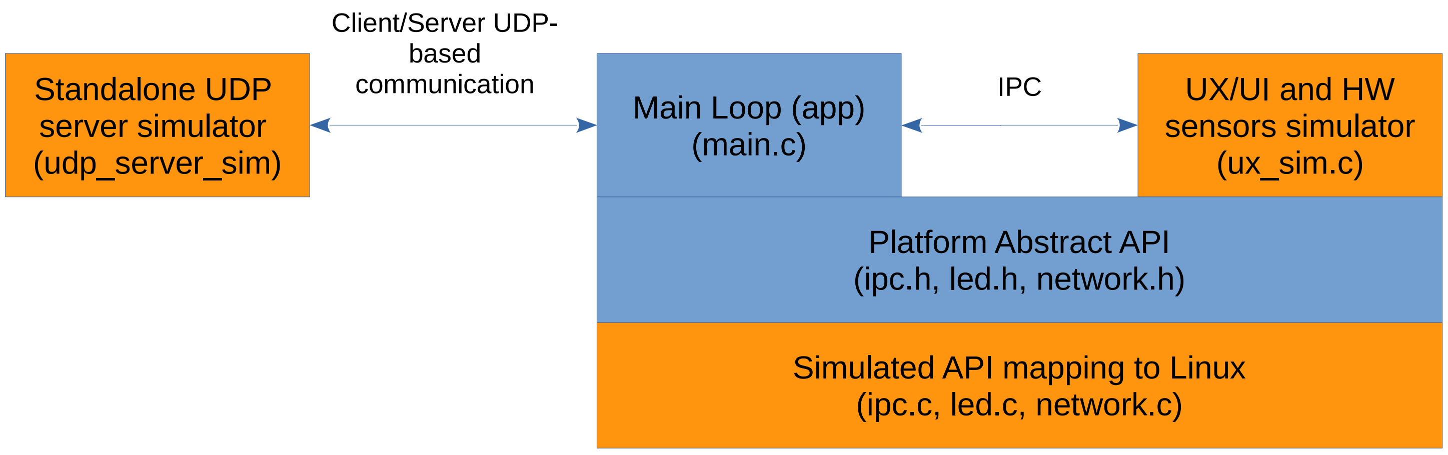
4.) Battery Operation

a.) While the battery voltage is <3.5V, the system shall be put in a non-functional state, meaning the white LED shall not illuminate and the Network Application shall not be executed, even if the button is pushed.

b.) While the battery voltage is <3.5V, the red LED shall blink at a rate of 2Hz with a 25% duty cycle.

c.) If the battery voltage returns to an operable level (>= 3.5V), the red LED shall stop blinking and operation shall resume as normal.

Design Overview


Client/Server UDP-based communication
Standalone UDP server simulator (udp_server_sim)
Main Loop (app) (main.c)
IPC
UX/UI and HW sensors simulator (ux_sim.c)
Platform Abstract API (ipc.h, led.h, network.h)
Simulated API mapping to Linux (ipc.c, led.c, network.c)

Build and run

Build

TestSample.zip contains:

- QT project TestSample.pro to build *TestSample* application
- Makefile to build all components: *TestSample*, *udp_server_sim* and *ux*

Run

Application are supposed to be started in certain order (to properly create queues and sockets)

1. Open terminal and start ./udp_server_sim
2. Open new terminal and start ./TestSample
3. Open 3rd terminal and start ./ux

Tests can be executed in the ./ux terminal window (see description below)

UX Simulator output

```
~/Qt5.5.0/Projects/TestSample$ ./ux
This module simulate UX for sample device
Usage:
space - toggle spring button: down/up
b     - toggle battery state: low/normal
c     - toggle charger state: in/out
e     - exit, end of the ux simulator
< > // Space
command bButtonDown now is down
b
command bLowPower now is low
c
command bChargerIn now is IN
c
command bChargerIn now is OUT
b
command bLowPower now is normal
< > // Space
command bButtonDown now is up
e

exited...
```

TestSample (main app) output

```
./TestSample
msgget: msgget succeeded: gMsgQueue = 1310720
IPC_BUTTON_DOWN
created network process pid = 42417
LED is OFF
blinking white started
localhost = 127.0.0.1
2 bytes sent to localhost, counter = 0
Ack from server: 00:00
2 bytes sent to localhost, counter = 1
Ack from server: 00:01
2 bytes sent to localhost, counter = 2
Ack from server: 00:02
2 bytes sent to localhost, counter = 3
Ack from server: 00:03
2 bytes sent to localhost, counter = 4
Ack from server: 00:04
IPC_LOW_POWER
waiting for child network process pid = 42417
ipc_retval = 4, 666
network task completed with err = 0
network process completed with status = 0
LED is OFF
blinking red started
IPC_CHARGER_ON
created network process pid = 42418
LED is OFF
blinking white started
localhost = 127.0.0.1
2 bytes sent to localhost, counter = 0
Ack from server: 00:00
2 bytes sent to localhost, counter = 1
Ack from server: 00:01
2 bytes sent to localhost, counter = 2
Ack from server: 00:02
2 bytes sent to localhost, counter = 3
Ack from server: 00:03
2 bytes sent to localhost, counter = 4
Ack from server: 00:04
2 bytes sent to localhost, counter = 5
Ack from server: 00:05
IPC_CHARGER_OFF
```

```
waiting for child network process pid = 42418
ipc_retval = 4, 666
network task completed with err = 0
network process completed with status = 0
LED is OFF
blinking red started
IPC_POWER_OK
created network process pid = 42419
LED is OFF
blinking white started
localhost = 127.0.0.1
2 bytes sent to localhost, counter = 0
Ack from server: 00:00
2 bytes sent to localhost, counter = 1
Ack from server: 00:01
2 bytes sent to localhost, counter = 2
Ack from server: 00:02
2 bytes sent to localhost, counter = 3
Ack from server: 00:03
IPC_BUTTON_UP
waiting for child network process pid = 42419
ipc_retval = 4, 666
network task completed with err = 0
network process completed with status = 0
LED is OFF
IPC_STOP_REQ
```

Limitations and Assumptions

The code contains comments about assumptions taken during development:

- LED driver/ctrl task is asynchronous process, accessible via message queue using commands defined in the led.h
- Power management sensor/module is asynchronous process, delivering (upon registration/subscription) notifications when power goes < 3.5V (once) and higher 3.5V (once).
- USB battery Charger detector is asynchronous process, delivering (upon registration/subscription) notifications when charger is plugged/unplugged.
- network.c module uses real BSD socket implementation for the communication with the server (in this project simulated with udp_server_sim); For test purposes the URL is used as “localhost” (vs. requested "test.ring.com") however it can be replaced with "test.ring.com" or any other.