# L_01

# Hardware I/O – LEDs Driving

# Table of Contents

# 1. Introduction

## 1.1 Purpose

*The purpose of this laboratory work is to get familiarized with the Embedded Artists AOAA board, the Eclipse CDT programming environment and with the Embedded C programming techniques for controlling the board hardware outputs.*

## 1.2 Objective

(1) Implementation of an embedded C application for driving RGB LEDs modules of the board, based on concurrent tasks and state machines.
(2) Update of the current template application with new functions for driving each LED of the RGB modules individually.
(3) Implementation of a counter based state change of the LEDs.

# 2. General Description

The Robot Controller Embedded System contains a hardware and a software subsystem. The hardware subsystem contains the AOAA Board and the J-Link Debugger (Fig. 1). The software subsystem contains the application implemented in Embedded C using Eclipse CDT programming environment.

## 2.1 Hardware subsystem

The Embedded Artists AOAA board is a dual-processor multi-purpose board having the following features:

- Main board with NXP LPC1769 microcontroller
- Secondary board with NXP LPC11C24 microcontroller
- External peripheral connections: General I/O, Ethernet link, XBee/WiFi connector (UART), Serial debugging interface, CAN interface with Secondary board, potentiometer via Analog-to-Digital converter input, user buttons via General I/O inputs

The Hardware subsystem communicates with the Programming Environment via J-Link Debugger connected through *JTAG* interface (Fig. 1).

The board can be powered up using the power supply connector (6-12V) or the USB connector.

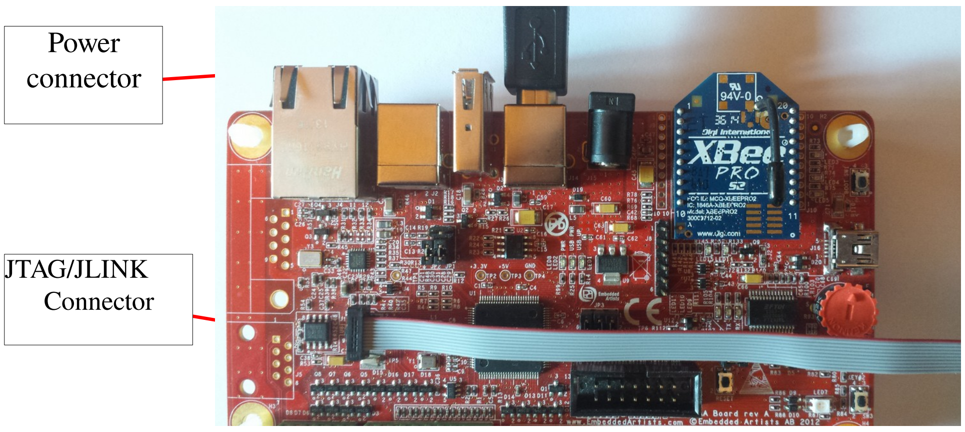
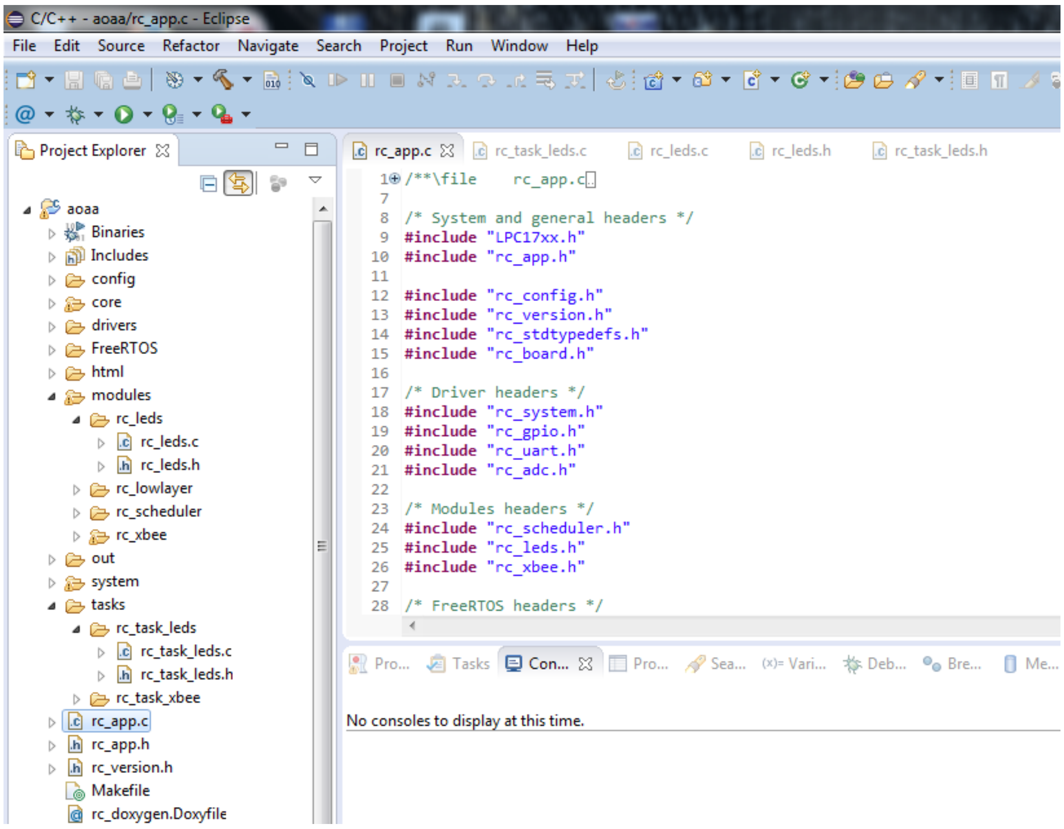


Figure 1. AOAA Board

## 2.2 Software subsystem

The software subsystem contains the following parts:

- Main application source and header: rc_app.c/.h
- *Config* folder: contains the header where Robot Controller macros and constants are defined
- *Core* and *Drivers* folders: contains the sources and headers which export the functions needed to access the processor peripherals (GPIO, ADC, UART, SPI, Timers, etc.)
- *FreeRTOS* folder: contains the operating system and main scheduler files
- *Html* folder: contains the user functions documentation generated with Doxygen.
- *Modules* folder: contains subfolders associated with the hardware components available on the board (RGB LEDs, Buttons, Potentiometer, XBee module, UART connectors, etc)
- *System* folder: contains the system files (interrupt vectors, linker scripts, processor definitions header, etc.)
- *Tasks* folder: contains subfolders associated with all the tasks running or available in the system (FreeRTOS type tasks associated with user functionalities).



```
aoaa
  Binaries
  Includes
  config
  core
  drivers
  FreeRTOS
  html
  modules
    rc_leds
      rc_leds.c
      rc_leds.h
    rc_lowlayer
    rc_scheduler
    rc_xbee
  out
  system
  tasks
    rc_task_leds
      rc_task_leds.c
      rc_task_leds.h
    rc_task_xbee
  rc_app.c
  rc_app.h
  rc_version.h
  Makefile
  rc_doxygen.Doxyfile
```

```
/**\file    rc_app.c

/* System and general headers */
#include "LPC17xx.h"
#include "rc_app.h"

#include "rc_config.h"
#include "rc_version.h"
#include "rc_stdtypedefs.h"
#include "rc_board.h"

/* Driver headers */
#include "rc_system.h"
#include "rc_gpio.h"
#include "rc_uart.h"
#include "rc_adc.h"

/* Modules headers */
#include "rc_scheduler.h"
#include "rc_leds.h"
#include "rc_xbee.h"

/* FreeRTOS headers */
```

No consoles to display at this time.

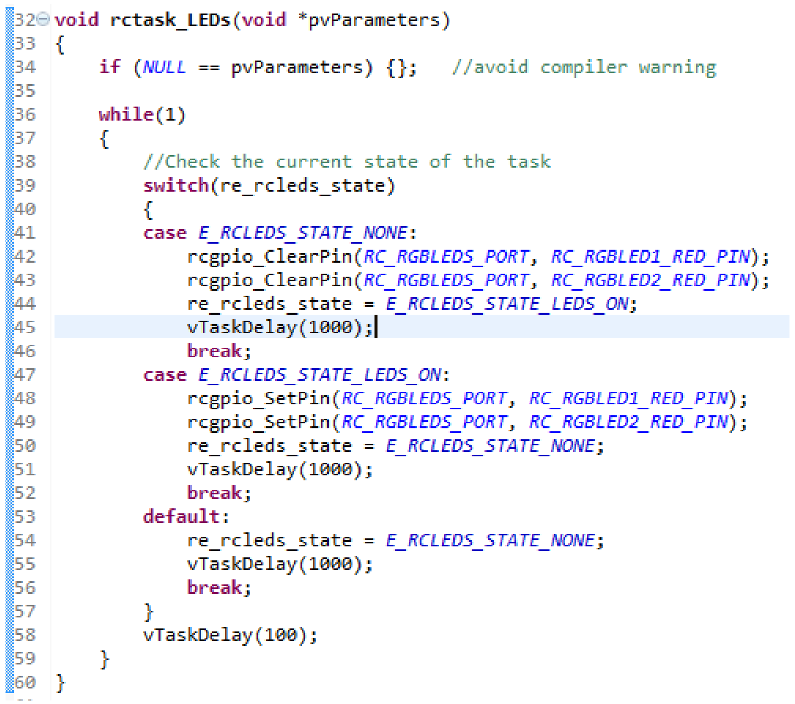
Figure 2. Software project structure

The board functionalities are driven by the FreeRTOS operating system scheduler. After the peripheral initialization, FreeRTOS starts the scheduler for each user enabled task. The context is shared between running tasks based on FreeRTOS switch context and task priorities (defined in *rc_app.c* main function, where tasks are initialized).

## 2.3 Software template description

The software template contains all the setups to run a standalone application for driving the board RGB LED modules.
There are two important sections in the code which need to be followed in order to understand the current functionality of the board:

- the LEDs task running continuously from *“tasks\rc_task_leds\rc_task_leds.c”*
- the application logic implemented in the LEDs task, which is based on a simple State-Machine.

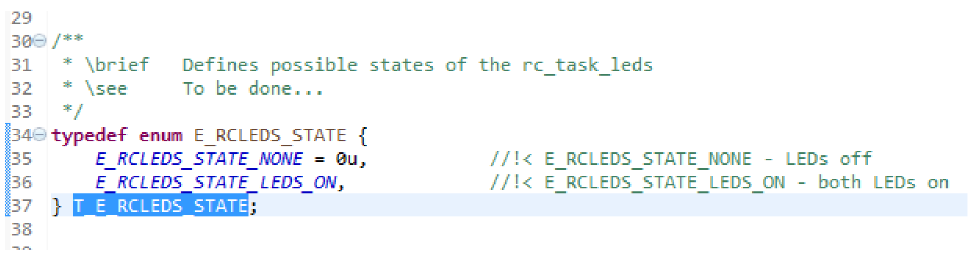


```c
void rctask_LEDs(void *pvParameters)
{
    if (NULL == pvParameters) {};   //avoid compiler warning

    while(1)
    {
        //Check the current state of the task
        switch(re_rcleds_state)
        {
        case E_RCLEDS_STATE_NONE:
            rcgpio_ClearPin(RC_RGBLEDS_PORT, RC_RGBLED1_RED_PIN);
            rcgpio_ClearPin(RC_RGBLEDS_PORT, RC_RGBLED2_RED_PIN);
            re_rcleds_state = E_RCLEDS_STATE_LEDS_ON;
            vTaskDelay(1000);
            break;
        case E_RCLEDS_STATE_LEDS_ON:
            rcgpio_SetPin(RC_RGBLEDS_PORT, RC_RGBLED1_RED_PIN);
            rcgpio_SetPin(RC_RGBLEDS_PORT, RC_RGBLED2_RED_PIN);
            re_rcleds_state = E_RCLEDS_STATE_NONE;
            vTaskDelay(1000);
            break;
        default:
            re_rcleds_state = E_RCLEDS_STATE_NONE;
            vTaskDelay(1000);
            break;
        }
        vTaskDelay(100);
    }
}
```

Figure 3. LEDs task and the State-Machine

In the *“rctask_LEDs”* function, the two defined states of the RGB LEDs modules are analyzed. The general state is saved in the static *“re_rcleds_state”* variable which is initially defined as *E_RCLEDS_STATE_NONE*. The two currently available states are hold by



```c

/**
 * \brief   Defines possible states of the rc_task_leds
 * \see     To be done...
 */
typedef enum E_RCLEDS_STATE {
    E_RCLEDS_STATE_NONE = 0u,           //!< E_RCLEDS_STATE_NONE - LEDs off
    E_RCLEDS_STATE_LEDS_ON,             //!< E_RCLEDS_STATE_LEDS_ON - both LEDs on
} T_E_RCLEDS_STATE;

```

Figure 4. LEDs states structure

the *T_E_RCLEDS_STATE* structure (Fig. 4), defined in the LEDs task header file.

In the continuous “*while*” loop of the task, the three possible cases are analyzed. For the NONE state of the LEDs, the application turns ON both RED LEDs of the RGB LEDs modules. This is done by calling the “*rcgpio_ClearPin*” function. The function requires two parameters: the LEDs port ID and the RED LED pin ID. The RGB LEDs modules are connected physically to the processor via the PORT2 hardware pins group as follows:

| ***LED*** | ***LED PORT ID*** | ***LED PIN ID*** |
|---|---|---|
| *RGB1 Red LED* | `RCLEDS_PORT` | `RCLEDS_RGB1_RED` |
| *RGB1 Blue LED* | `RCLEDS_PORT` | `RCLEDS_RGB1_BLUE` |
| *RGB1 Green LED* | `RCLEDS_PORT` | `RCLEDS_RGB1_GREEN` |
| *RGB2 Red LED* | `RCLEDS_PORT` | `RCLEDS_RGB2_RED` |
| *RGB2 Blue LED* | `RCLEDS_PORT` | `RCLEDS_RGB2_BLUE` |
| *RGB2 Green LED* | `RCLEDS_PORT` | `RCLEDS_RGB2_GREEN` |

After the LEDs turn ON action, the state of the LEDs modules is changed to “LEDS_ON”. Each state change ends with a delay of 1 second (1000 ms) set by the “*vTaskDelay*” function.

The second state is entered after the next task cycle, where the application turns OFF both RED LEDs by using “*rcgpio_SetPin*” function (same parameters as “*rcgpio_ClearPin*” function).

Overall, the main functionality of the board based on this application is to turn on and off the RED LEDs, with a pause of 1 second in each state.

## 3. Requirements

Based on the example described in the , update the application template with the following requirements:

(1) The application shall drive the two RGB LEDs modules using a State-Machine using the following defined states:

| ***State*** | ***State name*** | ***Condition*** | ***Period (ms)*** |
|---|---|---|---|
| *RGB1 Red Led On* | *E_RCLEDS_RED1_ON* | *Initial state, Other LEDs off* | *1000* |
| *RGB1 Blue Led On* | *E_RCLEDS_BLUE1_ON* | *Other LEDs off* | *2000* |
| *RGB2 Green Led On* | *E_RCLEDS_GREEN2_ON* | *Other LEDs off* | *5000 and switch to initial state* |
| *RGB1 Green Led On* | *E_RCLEDS_GREEN1_ON* | *Other LEDs off and RGB1 Red led turned ON 3 times* | *2000 and switch to initial state* |

The states name shall be defined in T_E_RCLEDS_STATE structure.
The task delay requested in “Period” column shall be applied after turning on the LEDs corresponding to the specified state.
The functional flow shall be:

1. Power ON -> *E_RCLEDS_RED1_ON*
2. *E_RCLEDS_RED1_ON -> E_RCLEDS_BLUE1_ON (or E_RCLEDS_GREEN1_ON if system has been in E_RCLEDS_RED1_ON minimum 3 times)*
3. *E_RCLEDS_BLUE1_ON -> E_RCLEDS_GREEN2_ON*
4. *E_RCLEDS_GREEN1_ON -> E_RCLEDS_RED1_ON*

(2) The application shall implement two new functions which are used for turning ON and turning OFF the LEDs. Both functions shall have an input parameter represented by the LEDs PIN ID. Considering that the LEDs hardware PORT is the same for both RGB LEDs modules (RCLEDS_PORT), the two newly defined functions shall represent a replacement for the functions “*rcgpio_ClearPin*” and “*rcgpio_SetPin*”.