Driving LEDs by GPIO: finally resolved!

january 17, 2013 by rtos.be leave a comment

Driving LEDs by GPIO: finally resolved!, 8.0 out of 10 based on 1 rating

Rating: 8.0/**10** (1 vote cast)

Introduction

Probably one of the most fascinating things you had to do in your early embedded career was controlling some LEDs by means of GPIO (General Purpose I/O). This – of course – is *so easy* i.e. until your project manager passes by:

> "I want this LED to blink a little faster"
> "Why is the green LED not blinking in sync with the red LED?"
> "This LED must blink 2 times faster than the other 2 LEDs."
> "I want this LED to turn on and off… just like an heart beat!"

Of course, next day the PM changes his/her mind. Frustration comes up… but again, abstraction is the key.

LED group and LED interface

If we look at things from the viewpoint of the 'high-level' application developer:

> the application developer wants to update all LED's of the same LED group *together* (i.e. in sync), and the application developer wants to configure the *mode* (i.e. the blinking pattern) of a LED.

The **mode of the LED** is crucial: it is *not* the state of the LED at one point in time (ON or OFF) but it is **a blinking pattern or a sequence of LED states**.
This brings us to 2 abstractions:

> a LED group in which you can put a set of LEDs and update them together,
> a LED abstraction which must be generic enough to deal with any kind of mode (a mode = a blinking pattern).

Let's start with the LED mode. This is no more than a sequence of LED states:

```
enum led_state {
    led_state_last = -1,
    led_off,
    led_on,
};
const enum led_state led_mode_on[] = {
    led_on,
    led_state_last
};
const enum led_state led_mode_off[] = {
    led_off,
    led_state_last
};
const enum led_state led_mode_blink_slow[] = {
    led_off,
    led_off,
    led_on,
    led_on,
    led_state_last
};
const enum led_state led_mode_blink_fast[] = {
    led_off,
    led_on,
    led_off,
    led_on,
    led_state_last
};
const enum led_state led_mode_heartbeat[] = {
    led_off,
    led_on,
    led_off,
    led_on,
    led_off,
    led_off,
    led_off,
    led_off,
    led_state_last
};
```

Any blinking pattern (= mode) can be created with the above abstraction. The LED mode then is used in a LED abstraction in which the *led_update()* function updates the led to the next led state:

```
typedef struct led {
    const PIN pin; //! cpu pin (will be configured as output gpio)
    const enum led_state* mode; //! an array of wanted led states
    unsigned char pos; //! the position in the array of wanted led
} LED;
```

```
/*!
* Initialize a led.
*/
void led_init(LED* p_led);

/*!
* Update a led to the next state.
*/
void led_update(LED* p_led);

/*!
* Change the mode (= array of wanted led states) of a led.
*/
void led_set_mode(LED* p_led, const enum led_state* led_mode);
```

Finally, LEDs are combined in a LED group:

```
//! definition of when a led table ends
#define LED_TABLE_END { .pin=PIN_INVALID }
static LED led_table[] = {
    {PIN1,led_mode_off,0},
    {PIN2,led_mode_on,0},
    {PIN3,led_mode_blink_slow,0},
    {PIN4,led_mode_blink_fast,0},
    LED_TABLE_END
};

/*!
* Initialize all leds of the group.
*/
void led_group_init(LED* led_table);

/*!
* Update all leds of the group.
*/
void led_group_update(LED* led_table);

/*!
* Set new mode for all leds of the group.
*/
void led_group_set_mode(LED* led_table, const enum led_state* new_
```

It is up to the application developer to decide how fast and in which context that the *led_group_update()* function must be called (the interface only provides mechanism and doesn't enforce any

policy). For example, if you want to update the state of the LEDs each 250ms then the application should call led_group_update() each 250ms.

Breadboard experiment

Let's play around a bit. We created a little test app with 4 LEDs in one LED group. When the user presses a button then the mode of all LEDs is changed. Watch our first video!

At application level the code is restricted to:

> a LED table *led_table* with 4 LEDs,
> one call to *led_group_init(led_table)*,
> periodic calls (e.g. each 100ms) to *led_group_update(led_table)*,
> some logic to change the mode of the LEDs when the button is pressed.

That's it.

Taking it to the next level

Of course, LED's are not only controlled by GPIO's e.g. dedicated LED drivers (IC's) exist. Again, from the viewpoint of the application developer, the implementation details (*how* a LED is driven) should be abstracted. This is possible by creating an abstract LED interface (for the C++ guys: look at it as an abstract base class) which has several implementations (LED_GPIO, LED_IC1…). Since we don't need this 'advanced' solution for the energy harvester project, we leave the details to you 🙂

Related Posts

Coming Soon
The embedded hierarchy — Part 1
Reverse engineering the bug of my Opel Zafira's volume knob (with movie!)
Debounced Buttons
MCU pin configuration, GPIOs and a word on software architecture

About rtos.be

rtos.be is a joined initiative from Gert Boddaert and Pieter Beyens.

More info: about us, our mission, contact us.