```c
/*
 * ETML-ES
 *
 * 2228_AlarmeFenetreOuverte
 *
 * Main NRF Récepteur
 *
 * Miguel Santos
 * 2023
 */
#include <zephyr/kernel.h>
#include <zephyr/drivers/gpio.h>
#include <zephyr/drivers/uart.h>

/* Main loop sleeping time in ms */
#define SLEEP_TIME_MS   100000

/* Action sent by uart on PIC32 */
#define ACTION1 0x0F

/* Devicetree nodes identifiers */
#define LED0_NODE DT_ALIAS(led0)
#define LED1_NODE DT_ALIAS(led1)
#define LED2_NODE DT_ALIAS(led2)
#define LED3_NODE DT_ALIAS(led3)
#define SW0_NODE DT_ALIAS(sw0)

#define UART_NODE DT_NODELABEL(uart0)

/* GPIO specifications */
static const struct gpio_dt_spec led0 = GPIO_DT_SPEC_GET(LED0_NODE, gpios);
static const struct gpio_dt_spec led1 = GPIO_DT_SPEC_GET(LED1_NODE, gpios);
static const struct gpio_dt_spec led2 = GPIO_DT_SPEC_GET(LED2_NODE, gpios);
static const struct gpio_dt_spec led3 = GPIO_DT_SPEC_GET(LED3_NODE, gpios);
static const struct gpio_dt_spec btn0 = GPIO_DT_SPEC_GET(SW0_NODE, gpios);

/* Infos about UART */
static const struct device *uart = DEVICE_DT_GET(UART_NODE);

/* Buffer to store incoming UART data*/
static uint8_t rx_buffer[10] = {0};

/* Infos about btn0 isr callback */
static struct gpio_callback btn0_cb_data;

/* Check if devices are available
 * if not, programm should be stop */
bool check_devices()
{
    bool device_error = false;

    if (!device_is_ready(led0.port)) {
        device_error = true;
    }

    if (!device_is_ready(led1.port)) {
        device_error = true;
    }

    if (!device_is_ready(led2.port)) {
        device_error = true;
    }

    if (!device_is_ready(led3.port)) {
        device_error = true;
    }

    if (!device_is_ready(btn0.port)) {
        device_error = true;
    }

    if (!device_is_ready(uart)) {
        device_error = true;
```

```
74      }
75
76      return device_error;
77  }
78
79  /* Configure pins to input or outputs */
80  bool configure_devices()
81  {
82      bool device_error = false;
83      int device_return;
84
85      device_return = gpio_pin_configure_dt(&led0, GPIO_OUTPUT_INACTIVE);
86      if (device_return < 0) {
87          device_error = true;
88      }
89
90      device_return = gpio_pin_configure_dt(&led1, GPIO_OUTPUT_INACTIVE);
91      if (device_return < 0) {
92          device_error = true;
93      }
94
95      device_return = gpio_pin_configure_dt(&led2, GPIO_OUTPUT_INACTIVE);
96      if (device_return < 0) {
97          device_error = true;
98      }
99
100     device_return = gpio_pin_configure_dt(&led3, GPIO_OUTPUT_INACTIVE);
101     if (device_return < 0) {
102         device_error = true;
103     }
104
105     device_return = gpio_pin_configure_dt(&btn0, GPIO_INPUT);
106     if (device_return < 0) {
107         device_error = true;
108     }
109
110     return device_error;
111 }
112
113 /* Callback function of btn0 ISR */
114 void btn0_cb(const struct device *dev, struct gpio_callback *cb, uint32_t pins)
115 {
116     if(gpio_pin_get_dt(&btn0))
117     {
118         gpio_pin_set_dt(&led0, true);
119     }
120     else
121     {
122         gpio_pin_set_dt(&led0, false);
123     }
124 }
125
126 /* Configure interrupt service routine */
127 bool btn0_configure_isr()
128 {
129     bool int_return;
130
131     /* Assign an interrupt to a pin and trigger edge */
132     int_return = gpio_pin_interrupt_configure_dt(&btn0, GPIO_INT_EDGE_BOTH);
133
134     /* Initialize the static struct gpio_callback variable */
135     gpio_init_callback(&btn0_cb_data, btn0_cb, BIT(btn0.pin));
136
137     /* Add the callback function by calling gpio_add_callback()   */
138     gpio_add_callback(btn0.port, &btn0_cb_data);
139
140     return int_return;
141 }
142
143 static void uart_cb(const struct device *dev, struct uart_event *evt, void *user_data)
144 {
145     switch (evt->type) {
146         case UART_TX_DONE:
```

```
            // do something
            break;
        case UART_TX_ABORTED:
            // do something
            break;
        case UART_RX_RDY:
            if(evt->data.rx.buf[evt->data.rx.offset] == ACTION1)
            {
                gpio_pin_toggle_dt(&led1);
            }
            break;
        case UART_RX_BUF_REQUEST:
            // do something
            break;
        case UART_RX_BUF_RELEASED:
            // do something
            break;
        case UART_RX_DISABLED:
            uart_rx_enable( uart, rx_buffer, sizeof(rx_buffer), 100);
            break;
        case UART_RX_STOPPED:
            // do something
            break;
        default:
            break;
    }
}

void main(void)
{
    bool error_return;

    /* Check if devices are configured right */
    error_return = check_devices();
    if(error_return){
        return;
    }

    /* Configure GPIO to input or output */
    error_return = configure_devices();
    if(error_return){
        return;
    }

    /* Configure UART callback routine */
    error_return = uart_callback_set(uart, uart_cb, NULL);
    if (error_return) {
        return;
    }

    /* Enable UART */
    uart_rx_enable(uart ,rx_buffer ,sizeof(rx_buffer) ,100);

    /* Configure interrupt related to btn0*/
    btn0_configure_isr();

    while (true) {
        k_msleep(SLEEP_TIME_MS);
    }
}

```