

```

1  /*
2  *  ETML-ES
3  *
4  *  2228_AlarmeFenetreOuverte
5  *
6  *  Main NRF Emetteur
7  *
8  *  Miguel Santos
9  *  2023
10 */
11
12 #include <zephyr/kernel.h>
13 #include <zephyr/drivers/gpio.h>
14
15
16 /* Main loop sleeping time in ms */
17 #define SLEEP_TIME_MS 100000
18
19 /* Devicetree nodes identifiers */
20 #define LED0_NODE DT_ALIAS(led0)
21 #define LED1_NODE DT_ALIAS(led1)
22 #define LED2_NODE DT_ALIAS(led2)
23 #define LED3_NODE DT_ALIAS(led3)
24
25 #define SW0_NODE DT_ALIAS(sw0)
26
27 #define VBAT_OK_NODE DT_ALIAS(vbat_ok)
28 #define MAG_OUT_NODE DT_ALIAS(mag_out)
29 #define MAG_EN_NODE DT_ALIAS(mag_en)
30
31
32 /* GPIO specifications */
33 static const struct gpio_dt_spec led0 = GPIO_DT_SPEC_GET(LED0_NODE, gpios);
34 static const struct gpio_dt_spec led1 = GPIO_DT_SPEC_GET(LED1_NODE, gpios);
35 static const struct gpio_dt_spec led2 = GPIO_DT_SPEC_GET(LED2_NODE, gpios);
36 static const struct gpio_dt_spec led3 = GPIO_DT_SPEC_GET(LED3_NODE, gpios);
37
38 static const struct gpio_dt_spec btn0 = GPIO_DT_SPEC_GET(SW0_NODE, gpios);
39
40 static const struct gpio_dt_spec batteryOk = GPIO_DT_SPEC_GET(VBAT_OK_NODE, gpios);
41 static const struct gpio_dt_spec sensorPin = GPIO_DT_SPEC_GET(MAG_OUT_NODE, gpios);
42 static const struct gpio_dt_spec sensorEnable = GPIO_DT_SPEC_GET(MAG_EN_NODE, gpios);
43
44
45 /* Define a variable of type static struct gpio_callback */
46 static struct gpio_callback btn0_cb_data;
47 static struct gpio_callback sensor_cb_data;
48
49
50 /* Check if devices are available
51  * if not, programm should be stop */
52 bool check_devices()
53 {
54     bool device_error = false;
55
56     if (!device_is_ready(led0.port)) {
57         device_error = true;
58     }
59
60     if (!device_is_ready(led1.port)) {
61         device_error = true;
62     }
63
64     if (!device_is_ready(led2.port)) {
65         device_error = true;
66     }
67
68     if (!device_is_ready(led3.port)) {
69         device_error = true;
70     }
71
72     if (!device_is_ready(btn0.port)) {
73         device_error = true;

```

```

74     }
75
76     if (!device_is_ready(batteryOk.port)) {
77         device_error = true;
78     }
79
80     if (!device_is_ready(sensorPin.port)) {
81         device_error = true;
82     }
83
84     if (!device_is_ready(sensorEnable.port)) {
85         device_error = true;
86     }
87
88     return device_error;
89 }
90
91 /* Configure pins to input or outputs */
92 bool configure_devices()
93 {
94     bool device_error = false;
95     int device_return;
96
97     device_return = gpio_pin_configure_dt(&led0, GPIO_OUTPUT_INACTIVE);
98     if (device_return < 0) {
99         device_error = true;
100     }
101
102     device_return = gpio_pin_configure_dt(&led1, GPIO_OUTPUT_INACTIVE);
103     if (device_return < 0) {
104         device_error = true;
105     }
106
107     device_return = gpio_pin_configure_dt(&led2, GPIO_OUTPUT_INACTIVE);
108     if (device_return < 0) {
109         device_error = true;
110     }
111
112     device_return = gpio_pin_configure_dt(&led3, GPIO_OUTPUT_INACTIVE);
113     if (device_return < 0) {
114         device_error = true;
115     }
116
117     device_return = gpio_pin_configure_dt(&btn0, GPIO_INPUT);
118     if (device_return < 0) {
119         device_error = true;
120     }
121
122     device_return = gpio_pin_configure_dt(&batteryOk, GPIO_INPUT);
123     if (device_return < 0) {
124         device_error = true;
125     }
126
127     device_return = gpio_pin_configure_dt(&sensorPin, GPIO_INPUT);
128     if (device_return < 0) {
129         device_error = true;
130     }
131
132     device_return = gpio_pin_configure_dt(&sensorEnable, GPIO_OUTPUT_INACTIVE);
133     if (device_return < 0) {
134         device_error = true;
135     }
136
137     return device_error;
138 }
139
140 void ISR_btn0(const struct device *dev, struct gpio_callback *cb, uint32_t pins)
141 {
142     if(gpio_pin_get_dt(&btn0))
143     {
144         gpio_pin_set_dt(&led0, true);
145     }
146     else

```

```

147     {
148         gpio_pin_set_dt(&led0, false);
149     }
150 }
151
152 void ISR_sensor(const struct device *dev, struct gpio_callback *cb, uint32_t pins)
153 {
154     if(gpio_pin_get_dt(&sensorPin))
155     {
156         gpio_pin_set_dt(&led3, false);
157     }
158     else
159     {
160         gpio_pin_set_dt(&led3, true);
161     }
162 }
163
164 bool ISR_btn0_configure()
165 {
166     bool int_return;
167
168     int_return = gpio_pin_interrupt_configure_dt(&btn0, GPIO_INT_EDGE_BOTH);
169
170     /* Initialize the static struct gpio_callback variable */
171     gpio_init_callback(&btn0_cb_data, ISR_btn0, BIT(btn0.pin));
172
173     /* Add the callback function by calling gpio_add_callback() */
174     gpio_add_callback(btn0.port, &btn0_cb_data);
175
176     return int_return;
177 }
178
179 bool ISR_sensor_configure()
180 {
181     bool int_return;
182
183     int_return = gpio_pin_interrupt_configure_dt(&sensorPin, GPIO_INT_EDGE_BOTH);
184
185     /* Initialize the static struct gpio_callback variable */
186     gpio_init_callback(&sensor_cb_data, ISR_sensor, BIT(sensorPin.pin));
187
188     /* Add the callback function by calling gpio_add_callback() */
189     gpio_add_callback(sensorPin.port, &sensor_cb_data);
190
191     return int_return;
192 }
193
194
195 void main(void)
196 {
197     //int gpio_return;
198     bool error_return;
199
200     error_return = check_devices();
201     if(error_return){
202         return;
203     }
204
205     error_return = configure_devices();
206     if(error_return){
207         return;
208     }
209
210     ISR_btn0_configure();
211     ISR_sensor_configure();
212
213     while (true) {
214         k_msleep(SLEEP_TIME_MS);
215     }
216 }
217

```