

```

1  #include "switch.h"
2
3  /// extended state -----
4  // "basic" state
5  typedef enum {SW_IDLE, SW_IN, SW_OUT} swm_state_t;
6  static swm_state_t g_swm_state;
7
8  // externally reachable objects
9  bool gb_swm_long_msg; // set (externally) to start a measurement
10 bool gb_swm_msg; // set when a measurement is completed
11
12 // parameter to rf_done_msg
13 bool volatile gb_swm_can_sleep; // this FSM can sleep
14
15 // hardware resources
16 static InterruptIn *g_swm;
17
18 static Timeout tout_20ms; // timeout // pulsacion de un segundo
19 static Timeout tout_1s;
20
21 // internal objects
22 static bool volatile gb_swm_initd; // true after call to rf_init()
23
24 static bool volatile swm_fall_evnt;
25 static bool volatile swm_rise_evnt;
26
27 static bool volatile tout_20ms_evnt;
28 static bool volatile tout_1s_evnt;
29
30 // end of extended state -----
31
32 // ISRs -----
33 // switch SWM ISR
34 static void swm_fall_isr(void) {
35     swm_fall_evnt = true;
36     gb_swm_can_sleep = false;
37 }
38
39 static void swm_rise_isr(void){
40     swm_rise_evnt = true;
41     gb_swm_can_sleep = false;
42 }
43
44
45 //timeout ISR
46 static void tout_20ms_isr (void) {
47     tout_20ms_evnt = true;
48     gb_swm_can_sleep = false;
49 }
50
51 static void tout_1s_isr (void) {
52     tout_1s_evnt = true;
53     gb_swm_can_sleep = false;
54 }
55
56
57 // end of ISRs -----
58
59 // FSM -----
60 void swm_fsm (void) {
61     if (gb_swm_initd) { // protect against calling rf_fsm() w/o a previous call to rf_init()
62         switch (g_swm_state) {
63
64             case SW_IN :
65                 swm_fall_evnt = false;
66                 swm_rise_evnt = false;
67                 tout_1s_evnt = false;
68
69                 if(tout_20ms_evnt){
70                     tout_20ms_evnt = false;
71
72                     if(OU == *g_swm){
73                         tout_1s.attach_us(tout_1s_isr, 1000000);
74                         g_swm_state = SW_OUT;
75
76                     }else{
77                         g_swm_state = SW_IDLE;
78                     }
79                 }
80                 break;
81
82             case SW_OUT :
83                 swm_fall_evnt = false;
84                 tout_20ms_evnt = false;

```

```

85
86     if(tout_ls_evnt){
87         tout_ls_evnt = false;
88         gb_swm_long_msg = true;
89         g_swm_state = SW_IDLE;
90
91     }else if(swm_rise_evnt){
92         swm_rise_evnt = false;
93         gb_swm_msg = true;
94         tout_ls.detach();
95         g_swm_state = SW_IDLE;
96
97     }else{ //nada
98     }
99     break;
100
101     default: //SW_IDLE
102         swm_rise_evnt = false;
103         tout_20ms_evnt = false;
104         tout_20ms_evnt = false;
105
106         if(swm_fall_evnt){
107             swm_fall_evnt = false;
108             tout_20ms.attach_us(tout_20ms_isr, 20000);
109             g_swm_state = SW_IN;
110         }
111
112         // -----
113     } // switch (rf_state)
114
115     __disable_irq();
116     if(!swm_fall_evnt && !tout_20ms_evnt && !swm_rise_evnt && !tout_ls_evnt ) {
117         gb_swm_can_sleep = true;
118     }
119     __enable_irq();
120 } // if (gb_rf_initd)
121 }
122 // end of FSM -----
123
124 // initialize FSM machinery -----
125 void swm_init(InterruptIn *swm){
126     if (!gb_swm_initd) {
127         gb_swm_initd = true; // protect against multiple calls to rf_init
128
129         // initialize state
130         g_swm_state = SW_IDLE;
131
132         // initial actions
133         tout_20ms_evnt = false;
134         tout_ls_evnt = false;
135
136         g_swm = swm;
137         g_swm->fall(swm_fall_isr);
138         g_swm->rise(swm_rise_isr);
139     }
140 }
141 // end of FSM initialization -----
142

```