```
#include "switch.h"
     /// extended state -----
 5
     typedef enum {SW_IDLE, SW_IN, SW_OUT} swm_state_t;
 6
     static swm_state_t g_swm_state;
            gb_swm_long_msg;  // set (externally) to start a measurement
gb_swm_msg;  // set when a measurement is completed
 9
     bool
10
     bool
11
     12
13
14
15
16
     static InterruptIn *g_swm;
17
                                              // timeout // <u>pulsacion</u> de un segundo
     static Timeout
                          tout 20ms;
18
                       tout_1s;
19
     static Timeout
20
21
22
     static bool volatile gb_swm_initd;
                                                    // true after call to rf init()
23
     static bool volatile swm_fall_evnt;
static bool volatile swm_rise_evnt;
24
25
26
27
     static bool volatile tout 20ms evnt;
28
     static bool volatile tout_1s_evnt;
29
30
31
32
       // swhtch SWM ISR
34
     static void swm_fall_isr(void) {
     swm_fall_evnt = true;
3.5
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
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;
       gb_swm_can_sleep = false;
53
54
5.5
56
57
58
59
     void swm_fsm (void) {
60
       if (gb swm initd) { // protect against calling rf fsm() w/o a previous call to rf init()
61
        switch (g_swm_state) {
62
63
64
           case SW_IN :
6.5
              swm_fall_evnt = false;
66
              swm rise evnt = false;
              tout_1s_evnt = false;
67
68
69
              if(tout_20ms_evnt) {
70
                 tout_20ms_evnt = false;
71
                 if(0U == *g swm){
72
73
                   tout_1s.attach_us(tout_1s_isr, 1000000);
74
                   g_swm_state = SW_OUT;
75
76
                 }else{
                   g_swm_state = SW IDLE;
77
78
79
80
              break;
81
82
           case SW OUT :
              swm fall evnt = false;
83
               tout_20ms_evnt = false;
84
```

```
8.5
86
                if(tout_1s_evnt){
87
                    tout 1\overline{s} evnt = false;
88
                     gb_swm_long_msg = true;
                     g_swm_state = SW IDLE;
89
90
91
                  }else if(swm_rise_evnt) {
92
                     swm rise evnt = false;
93
                      gb swm msg = true;
94
                      tout 1s.detach();
95
                      g_swm_state = SW_IDLE;
96
97
                   }else{ //nada
98
99
                   break;
100
101
               default:
102
                  swm rise evnt = false;
                  tout_20ms_evnt = false;
103
                 tout_20ms_evnt = false;
104
105
106
                 if(swm_fall_evnt) {
107
                   swm \overline{f}all \overline{e}vnt = false;
                   tout_20ms.attach_us(tout_20ms_isr, 20000);
g_swm_state = SW_IN;
108
109
110
111
112
       } // switch (rf_state)
113
114
          disable_irq();
115
116
        if(!swm fall evnt && !tout 20ms evnt && !swm rise evnt && !tout 1s evnt ) {
117
           gb_swm_can_sleep = true;
118
119
          __enable_irq();
        } // if (ab_rf_initd)
120
121
122
123
124
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
130
         g_swm_state = SW_IDLE;
131
132
          tout_20ms_evnt = false;
133
134
          tout_1s_evnt = false;
135
         g_swm = swm;
136
137
          g_swm->fall(swm_fall isr);
138
         g_swm->rise(swm_rise_isr);
139
140
141
142
```