```
#include "switch.h"

/// extended state ------------------------------------------------------
  // "basic" state
typedef enum {SW_IDLE, SW_IN, SW_OUT} swm_state_t;
static swm_state_t g_swm_state;

  // externally reachable objects
bool              gb_swm_long_msg;    // set (externally) to start a measurement
bool              gb_swm_msg;     // set when a measurement is completed

    // parameter to rf_done_msg
bool volatile     gb_swm_can_sleep;    // this FSM can sleep

  // hardware resources
static InterruptIn    *g_swm;

static Timeout         tout_20ms;        // timeout // pulsacion de un segundo
static Timeout         tout_1s;

  // internal objects
static bool volatile  gb_swm_initd;             // true after call to rf_init()

static bool volatile  swm_fall_evnt;
static bool volatile  swm_rise_evnt;

static bool volatile tout_20ms_evnt;
static bool volatile tout_1s_evnt;

// end of extended state  ----------------------------------------------

// ISRs ----------------------------------------------------------------
  // swhtch SWM ISR
static void swm_fall_isr(void) {
  swm_fall_evnt = true;
  gb_swm_can_sleep = false;
}

static void swm_rise_isr(void){
  swm_rise_evnt = true;
  gb_swm_can_sleep  = false;
}


//timeout ISR
static void tout_20ms_isr (void) {
  tout_20ms_evnt = true;
  gb_swm_can_sleep = false;
}

static void tout_1s_isr (void) {
  tout_1s_evnt = true;
  gb_swm_can_sleep = false;
}


// end of ISRs  --------------------------------------------------------

// FSM  ----------------------------------------------------------------
void swm_fsm (void) {
  if (gb_swm_initd) {  // protect against calling rf_fsm() w/o a previous call to rf_init()
    switch (g_swm_state) {

      case SW_IN :
         swm_fall_evnt = false;
         swm_rise_evnt = false;
         tout_1s_evnt = false;

         if(tout_20ms_evnt){
            tout_20ms_evnt = false;

            if(0U == *g_swm){
              tout_1s.attach_us(tout_1s_isr, 1000000);
              g_swm_state = SW_OUT;

            }else{
              g_swm_state = SW_IDLE;
            }
         }
         break;

      case SW_OUT :
          swm_fall_evnt = false;
          tout_20ms_evnt = false;
```

```
85
86              if(tout_1s_evnt){
87                  tout_1s_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_1s.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_1s_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_1s_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
```