```
#include "range_finder.h"

// extended state -------------------------------------------------------
  // "basic" state
typedef enum {RF_IDLE, RF_TRG, RF_WAIT, RF_MEAS} rf_state_t;
static rf_state_t g_rf_state;

  // externally reachable objects
bool              gb_rf_start_msg;     // set (externally) to start a measurement
bool              gb_rf_done_msg;      // set when a measurement is completed
    // parameter to rf_done_msg
int32_t           g_rf_range_cm;       // measured range in cm
bool volatile     gb_rf_can_sleep;     // this FSM can sleep

  // hardware resources
static DigitalOut     *gp_rf_trigger;
static InterruptIn    *gp_rf_echo;
static Timeout        g_rf_to;         // timeout
static Timer          g_rf_tmr;        // measurement timer

  // internal objects
static bool volatile  gb_rf_initd;             // true after call to rf_init()
static bool volatile  gb_rf_echo_rise_evnt;    // rise on echo event
static bool volatile  gb_rf_echo_fall_evnt;    // fall on echo event
static bool volatile  gb_rf_to_evnt;           // timeout elapsed event


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

// ISRs -----------------------------------------------------------------
  // echo rise ISR
static void rf_echo_rise_isr(void) {
  gb_rf_echo_rise_evnt = true;
  gb_rf_can_sleep = false;
}

  // echo fall isr
static void rf_echo_fall_isr(void) {
  gb_rf_echo_fall_evnt = true;
  gb_rf_can_sleep = false;
}

  // timeout ISR
static void rf_to_isr(void) {
  gb_rf_to_evnt = true;
  gb_rf_can_sleep = false;
}


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

// FSM  -----------------------------------------------------------------
void rf_fsm (void) {
  if (gb_rf_initd) {  // protect against calling rf_fsm() w/o a previous call to rf_init()
    switch (g_rf_state) {
      // complete this code to achieve the FSM functionality  ++++++++++
       case RF_IDLE :

        if(gb_rf_start_msg){
          gb_rf_start_msg = false;
           *gp_rf_trigger = 1;
          g_rf_to.attach_us(rf_to_isr, 1000);
           g_rf_state = RF_TRG;

        }
       break;

      case RF_TRG :
        gb_rf_start_msg = false;

        if(gb_rf_to_evnt){
           gb_rf_start_msg = false;
          *gp_rf_trigger = 0;

           g_rf_tmr.stop();
          g_rf_tmr.reset();

          g_rf_state = RF_WAIT;

        }

       break;

      case RF_WAIT :
```

```
85
86                if(gb_rf_echo_rise_evnt){
87                  gb_rf_echo_rise_evnt = false;
88                  gb_rf_start_msg = true;
89                  gb_rf_done_msg = false;
90                  g_rf_tmr.start();
91                  g_rf_state = RF_MEAS;
92                }
93
94           break;
95
96          case RF_MEAS :
97
98              if(gb_rf_echo_fall_evnt){
99                gb_rf_echo_fall_evnt = false;
100                gb_rf_start_msg = false;
101                 gb_rf_done_msg = true;
102                 g_rf_tmr.stop();
103                 g_rf_range_cm = g_rf_tmr.read_us()/58;
104                  g_rf_state = RF_IDLE;
105             }
106
107           break;
108
109
110         //  ------------------------------------------------------------
111       } // switch (rf_state)
112
113
114       __disable_irq();
115       if(!gb_rf_echo_rise_evnt && !gb_rf_echo_fall_evnt && !gb_rf_to_evnt ) {
116         gb_rf_can_sleep = true;
117       }
118       __enable_irq();
119     } // if (gb_rf_initd)
120   }
121   // end of FSM ----------------------------------------------------------
122
123   // initialize FSM machinery  -------------------------------------------
124   void rf_init (DigitalOut *trigger, InterruptIn *echo) {
125     if (!gb_rf_initd) {
126       gb_rf_initd = true;    // protect against multiple calls to rf_init
127
128       // initialize state
129       g_rf_state = RF_IDLE;
130       gb_rf_echo_rise_evnt = false;
131       gb_rf_echo_fall_evnt = false;
132       g_rf_to.detach();
133       gb_rf_to_evnt = false;
134
135       // initial actions
136       gp_rf_trigger = trigger;              // save pointers to the pins
137       gp_rf_echo = echo;
138
139       *gp_rf_trigger = 0;                   // drive trigger
140       gp_rf_echo->rise(rf_echo_rise_isr);   // register echo ISRs
141       gp_rf_echo->fall(rf_echo_fall_isr);
142     }
143   }
144   // end of FSM initialization  ------------------------------------------
145
```