```c
/* Includes ------------------------------------------------------------------*/
#include "stm8s.h"
#include "stm8s_io.h"
#include "stm8s_timer.h"
#include "stm8s_analog.h"
#include "stm8s_counter.h"

#include "sensor_control.h"
#include "water_control.h"
#include "power_control.h"


#include <stdio.h>

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
uint16_t control = 0;

/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
/* Public  functions ---------------------------------------------------------*/

/**
  * @brief  Main program.
  * @param  None
  * @retval None
  */
void main(void){

   CLK->CKDIVR = 0;

   enableInterrupts();

   ioInit     ();
   timerInit  ();
   counterInit();


   analogSetBuffer(&control);

   while (1){

     if(sensorGetStatus()){

       powerControl(control);
       waterControl(ENABLE);

      }else{

       powerOff();

       waterControl(DISABLE);

      }

     sensorControl();

     if(timeBase1ms()){

       sensorRead();

       analogRun();

      }
    }
  }
```

```
/*****************************************************************
 * Biblioteca de Controle de Potência                            *
 *****************************************************************/



/*****************************************************************/

#include "stm8s.h"

/*****************************************************************
 * Funções e Procedimentos                                       *
 *****************************************************************/
/*****************************************************************
 * powerControl(rate);                                           *
 *                                                               *
 * Procedimento de controle de Potência;                         *
 *                                                               *
 * uint8_t -> quantidade de cliclos ativos;                      *
 *                                                               *
 * void -> não retorna valor;                                    *
 *****************************************************************/
void powerControl(uint16_t rate);

/*****************************************************************
 * powerOff();                                                   *
 *                                                               *
 * Procedimento de corte de energia;                             *
 *                                                               *
 * void -> não recebe parâmetros;                                *
 *                                                               *
 * void -> não retorna valor;                                    *
 *****************************************************************/
void powerOff(void);

/*****************************************************************/
```

```
/*****************************************************************
 * Biblioteca de Leitura do Sensor de Movimento                  *
 *****************************************************************/


/*****************************************************************/

#include "stm8s.h"

enum sensor{

   SENSOR_IDLE,
   SENSOR_READING

  };

/*****************************************************************
 * Funções e Procedimentos                                       *
 *****************************************************************/
/*****************************************************************
 * sensorControl();                                              *
 *                                                               *
 * Procedimento de controle do Sensor de Movimento;              *
 *                                                               *
 * void -> não recebe parâmetros;                                *
 *                                                               *
 * void -> não retorna valor;                                    *
 *****************************************************************/
void sensorControl(void);

/*****************************************************************
 * sensorRead();                                                 *
 *                                                               *
 * Procedimento de leitura do status do sensor;                  *
 *                                                               *
 * void -> não recebe parâmetros;                                *
 *                                                               *
 * void -> não retorna valor;                                    *
 *****************************************************************/
void sensorRead(void);

/*****************************************************************
 * sensorGetStatus();                                            *
 *                                                               *
 * Procedimento de leitura do status do sensor;                  *
 *                                                               *
 * void -> não recebe parâmetros;                                *
 *                                                               *
 * _bool -> retorna true se foi detectado movimento;             *
 *****************************************************************/
bool sensorGetStatus(void);

/*****************************************************************/
```

```
1   /*************************************************************
2    * Biblioteca de Contagem                                    *
3    *************************************************************/
4
5   /*************************************************************/
6
7   #include "stm8s.h"
8   #include "stm8s_io.h"
9
10
11  /*************************************************************
12   * Funções e Procedimentos                                   *
    *
13   *************************************************************/
14  /*************************************************************
15   * void ioInit(void);                                        *
16   *                                                           *
17   * Inicialização dos IO's                                    *
18   *                                                           *
19   *************************************************************/
20   void ioInit(void){
21
22  // GPIO_Init(ZERO_CROSSING, GPIO_MODE_IN_FL_NO_IT);
23     GPIO_Init(SENSOR,         GPIO_MODE_IN_FL_NO_IT);
24     GPIO_Init(POWER_CONTROL, GPIO_MODE_OUT_OD_LOW_SLOW);
25     GPIO_Init(BUZZER,         GPIO_MODE_OUT_PP_LOW_SLOW);
26     GPIO_Init(WATER,          GPIO_MODE_OUT_PP_LOW_SLOW);
27
28    }
29
```

```
/*****************************************************************
 * Biblioteca de Temporização                                    *
 *****************************************************************/
 /*****************************************************************/

#include "stm8s.h"
#include "stm8s_timer.h"
#include "jiga_park.h"

uint8_t i           = 0;

bool    timeBase_1ms = FALSE;
bool    timeBase_1us = FALSE;

/*****************************************************************
 * Vetor de Timers                                               *
 *****************************************************************/
timer timerUnit[TIMER_QTY];

/*****************************************************************
 * Funções e Procedimentos                                       *
*                                                                 *
 *****************************************************************/
/*****************************************************************
 * void timerInit(void);                                         *
 *                                                               *
 * Configurações do Timer 4:                                     *
 *                                                               *
 * Prescaler = 1                                                 *
 * Overflow  = 16                                                *
 *                                                               *
 * T = PRESCALER * OVERFLOW / FREQUENCIA DE CLOCK                *
 *                                                               *
 * T = 64 * 250 / 16.000.000 = 1ms                               *
*                                                                 *
 *****************************************************************/
void timerInit(void){

  TIM4_DeInit      ();
  TIM4_TimeBaseInit(TIM4_PRESCALER_64, 250);
  TIM4_ITConfig    (TIM4_IT_UPDATE, ENABLE);
  TIM4_Cmd         (ENABLE);

  TIM3_DeInit      ();
  TIM3_TimeBaseInit(TIM3_PRESCALER_16,  50);
  TIM3_ITConfig    (TIM3_IT_UPDATE, ENABLE);
  TIM3_Cmd         (ENABLE);



}

/*****************************************************************
 * timerEnable (unit, enableCount);                              *
 *                                                               *
 * Controle de habilitação do timer                              *
 *                                                               *
 * uint8_t unit        -> índice do timer a ser utilizado;       *
 * _Bool   enableCount  -> valor de hab/inib do timer;           *
 *                                                               *
 * void -> não retorna valor;                                    *
 *****************************************************************/
void timerEnable(uint8_t unit, _Bool enableTimer){

  if(enableTimer){

    timerUnit[unit].enable = 1;

  }else{

    timerUnit[unit].enable = 0;

```

```
71      }
72
73    }
74
75    /*************************************************************
76     * timerReset (unit);                                        *
77     *                                                           *
78     * Reset do valor de contagem do timer                       *
79     *                                                           *
80     * uint8_t unit        -> índice do timer a ser utilizado;   *
81     *                                                           *
82     * void -> não retorna valor;                                *
83     *************************************************************/
84    bool timerGetOverflow(uint8_t unit){
85
86      if(timerUnit[unit].overflow == 1){
87
88        return 1;
89
90      }else{
91
92        return timerUnit[unit].overflow;
93
94      }
95
96    }
97
98    /*************************************************************
99     * timerGetCount (unit);                                     *
100    *                                                           *
101    * Leitura do valor de contagem do timer                     *
102    *                                                           *
103    * uint8_t unit        -> índice do timer a ser utilizado;   *
104    *                                                           *
105    * uint16_t -> contagem do timer;                            *
106    *************************************************************/
107   uint16_t timerGetCount(uint8_t unit){
108
109     return timerUnit[unit].count;
110
111   }
112
113   /*************************************************************
114    * timerSetOverflowValue (unit, maxValue);                   *
115    *                                                           *
116    * Define o valor de overflow do timer                       *
117    *                                                           *
118    * uint8_t unit     -> índice do timer a ser utilizado;      *
119    * int     maxvalue -> valor de overflow do timer;           *
120    *                                                           *
121    * void -> não retorna valor;                                *
122    *************************************************************/
123   void timerSetOverflowValue(uint8_t unit, int maxValue){
124
125     timerUnit[unit].maxCount = maxValue;
126
127   }
128
129   /*************************************************************
130    * timerGetOverflow (unit);                                  *
131    *                                                           *
132    * Verifica se ocorreu overflow no timer                     *
133    *                                                           *
134    * uint8_t unit        -> índice do timer a ser utilizado;   *
135    *                                                           *
136    * _Bool -> retorna o valor de overflow do timer;            *
137    *************************************************************/
138   void timerReset(uint8_t unit){
139
140     timerUnit[unit].count    = 0;
141     timerUnit[unit].overflow = 0;
142
```

```
143   }
144
145   /*************************************************************
146    * timerDelay (delay);                                       *
147    *                                                           *
148    * Função de delay                                           *
149    *                                                           *
150    * uint8_t delay -> valor de duração do delay, em ms;        *
151    *                                                           *
152    * Bool -> retorna true enquanto não houve overflow no timer;*
153    *************************************************************/
154   bool timerDelay(uint16_t delay){
155
156     if(timerUnit[0].count == 0){
157
158       timerSetOverflowValue(0, delay);
159       timerReset           (0);
160       timerEnable          (0, ENABLE);
161
162     }
163
164     if(timerGetOverflow(0)){
165
166       timerEnable(0, DISABLE);
167       timerReset (0);
168
169       return TRUE;
170
171     }else{
172
173       return FALSE;
174
175     }
176   }
177
178   /*************************************************************
179    * bool timeBase1ms();                                       *
180    *                                                           *
181    * Verificação de base de tempo de 1ms                       *
182    *                                                           *
183    * Bool -> retorna true após um intervalo de 1ms;            *
184    *************************************************************/
185   bool timeBase1ms(void){
186
187     if(timeBase_1ms){
188
189       timeBase_1ms = FALSE;
190
191       return TRUE;
192
193     }else{
194
195       return FALSE;
196
197     }
198
199   }
200
201   /*************************************************************
202    * bool timeBase1us();                                       *
203    *                                                           *
204    * Verificação de base de tempo de 1us                       *
205    *                                                           *
206    * Bool -> retorna true após um intervalo de 1us;            *
207    *************************************************************/
208   bool timeBase1us(void){
209
210     if(timeBase_1us){
211
212       timeBase_1us = FALSE;
213
214       return TRUE;
```

```
215
216       }else{
217
218         return FALSE;
219
220       }
221
222     }
223
224
225     /*****************************************************************
226       * Interrupção do TIM4 (Timer físico do controlador;           *
227     *****************************************************************/
228     #ifdef STM8S903
229     /**
230       * @brief Timer6 Update/Overflow/Trigger Interrupt routine.
231       * @param  None
232       * @retval None
233       */
234     INTERRUPT_HANDLER(TIM6_UPD_OVF_TRG_IRQHandler, 23)
235      {
236       /* In order to detect unexpected events during development,
237          it is recommended to set a breakpoint on the following instruction.
238       */
239      }
240     #else /*STM8S208, STM8S207, STM8S105 or STM8S103 or STM8AF52Ax or STM8AF62Ax or
        STM8AF626x */
241     /**
242       * @brief Timer4 Update/Overflow Interrupt routine.
243       * @param  None
244       * @retval None
245       */
246     INTERRUPT_HANDLER(TIM4_UPD_OVF_IRQHandler, 23)
247       {
248
249         TIM4_ClearITPendingBit(TIM4_IT_UPDATE);
250
251         timeBase_1ms = TRUE;
252
253         for(i = 0; i < TIMER_QTY; i++){
254
255           if(timerUnit[i].enable == 1){
256
257             if(timerUnit[i].count >= timerUnit[i].maxCount){
258
259               timerUnit[i].overflow = 1;
260
261             }
262
263             timerUnit[i].count++;
264
265           }
266         }
267       }
268
269     #endif /*STM8S903*/
270
271     INTERRUPT_HANDLER(TIM3_UPD_OVF_BRK_IRQHandler, 15){
272
273        TIM3_ClearITPendingBit(TIM3_IT_UPDATE);
274
275         timeBase_1us = TRUE;
276
277       }
278
279
280
281
282
283
```

```
/*****************************************************************
 * Biblioteca de Aquisição Analógica                             *
 *****************************************************************/


#include "stm8s.h"
#include "stm8s_adc1.h"
#include "stm8s_analog.h"

 uint8_t sample     = 0;
 uint8_t channel    = 0;
 uint8_t dataStatus = DATA_UNAVAILABLE;

uint16_t  adcValue      = 0;

uint16_t *analog_buffer = (uint16_t *)0;

uint16_t  table[SAMPLE_QTY];



/*****************************************************************
 * ADC1_Enable(channel)                                          *
 *                                                               *
 * Configurações do Conversor Analógico/Digital 1                *
 *                                                               *
 * Modo de Conversão contínua                                    *
 * Bits alinhados à direita                                      *
 *                                                               *
 * _ _ _ _ _ _ D9 D8 - D7 D6 D5 D4 D3 D2 D1 D0                   *
 *                                                               *
 *****************************************************************/
void ADC1_Enable(uint8_t ch){

    ADC1_ConversionConfig(ADC1_CONVERSIONMODE_SINGLE, ch, ADC1_ALIGN_RIGHT);
    ADC1_Cmd              (ENABLE);
    ADC1_StartConversion ();

  }

/*****************************************************************
 * analogSetBuffer(&buffer_addr);                                *
 *                                                               *
 * Função para seleção do endereço do buffer de leitura;         *
 *                                                               *
 * uint16_t *buffer_addr -> endereço para o buffer               *
 *                                                               *
 * void -> não retorna valor;                                    *
 *****************************************************************/
void analogSetBuffer(uint16_t * buffer_addr){

   analog_buffer = buffer_addr;

  }

/*****************************************************************
 * analog_DataAvailable():                                       *
 *                                                               *
 * Verifica se a lista de valores do ADC foi preenchida          *
 *                                                               *
 * bool -> retorna true se a lista está completa, retorna        *
 *         se a lista ainda estiver sendo preenchida             *
 *                                                               *
 *****************************************************************/
 bool analogDataAvailable(void){

   if(dataStatus){

     dataStatus = DATA_UNAVAILABLE;

     return TRUE;

```

```
   73        }else{
   74
   75          return FALSE;
   76
   77        }
   78
   79      }
   80
   81    /*************************************************************
   82     * analogRun();                                              *
   83     *                                                           *
   84     * Configurações iniciais do ADC                             *
   85     *                                                           *
   86     * void -> não retorna valor                                 *
   87     *************************************************************/
   88     void analogRun(void){
   89
   90       signed int aux;
   91
   92       table[sample] = (uint32_t)ADC1_GetConversionValue() * (uint16_t)5000 / (uint16_t)1023;
   93
   94       if(sample > 0){
   95
   96         aux = table[sample] - table[sample-1];
   97
   98         if((aux > NOISE_REJECTION)||(aux < (-NOISE_REJECTION))){
   99
  100           channel    = 0;
  101           sample     = 0;
  102           dataStatus = DATA_UNAVAILABLE;
  103
  104           return;
  105
  106          }
  107
  108        }
  109
  110       sample++;
  111
  112       if(sample == SAMPLE_QTY){
  113
  114         analog_buffer[channel] = table[sample - 1];
  115
  116         sample = 0;
  117
  118         channel++;
  119
  120        }
  121
  122       if(channel == CHANNEL_QTY){
  123
  124         channel = 0;
  125
  126         dataStatus = DATA_AVAILABLE;
  127
  128        }
  129
  130       ADC1_Enable(channel);
  131
  132      }
  133
  134
  135
  136
  137
  138
```

```
/*****************************************************************
 * Biblioteca de Contagem                                        *
 *****************************************************************/

/*****************************************************************/

#include "stm8s.h"
#include "stm8s_counter.h"

uint8_t  count      = 0;
uint8_t  limitCount = 0;


/*****************************************************************
 * Funções e Procedimentos                                       *
 *****************************************************************/
/*****************************************************************
 * void counterInit(void);                                       *
 *                                                               *
 * Inicialização do TIMER1 com clock externo                     *
 *                                                               *
 *****************************************************************/
 void counterInit(void){

   TIM1_DeInit();

   TIM1_TimeBaseInit(99, TIM1_COUNTERMODE_UP, 3200, 0);

   TIM1_ICInit(TIM1_CHANNEL_2, TIM1_ICPOLARITY_RISING, TIM1_ICSELECTION_DIRECTTI, TIM1_ICPSC_DIV1, 0x00);

  }

/*****************************************************************
 * void setMaxCount(maxCount);                                   *
 *                                                               *
 * uint16_t maxValue -> seleção do maior valor de contagem;      *
 *                                                               *
 * void -> não retorna valor;                                    *
 *****************************************************************/
void setMaxCount(uint8_t maxCount){

   limitCount = maxCount;

  }

/*****************************************************************
 * uint8_t counterGetValue();                                    *
 *                                                               *
 * void -> não recebe parâmetros;                                *
 *                                                               *
 * uint8_t -> retorna o valor da contagem;                       *
 *****************************************************************/
uint8_t counterGetValue(void){

   return count;

  }

/*****************************************************************
 * void counterEnable();                                         *
 *                                                               *
 * void -> não retorna valor;                                    *
 *                                                               *
 *****************************************************************/
void counterEnable(void){

   TIM1_ITConfig(TIM1_IT_CC2,    ENABLE);
   TIM1_ITConfig(TIM1_IT_UPDATE, ENABLE);

   TIM1_Cmd(ENABLE);
```

```
71
72      }
73
74    /*************************************************************
75      * Interrupção do TIM1 (Capture e Compare);                  *
76     *************************************************************/
77     INTERRUPT_HANDLER(TIM1_CAP_COM_IRQHandler, 12){
78
79       TIM1_ClearITPendingBit(TIM1_IT_CC2);
80
81       count++;
82
83       if(count > (limitCount)){
84
85         count = 1;
86
87        }
88
89       TIM1_SetCounter(0);
90
91      }
92
93    /*************************************************************
94      * Interrupção do TIM1 (Overflow);                           *
95     *************************************************************/
96     INTERRUPT_HANDLER(TIM1_UPD_OVF_TRG_BRK_IRQHandler, 11){
97
98       TIM1_ClearITPendingBit(TIM1_IT_UPDATE);
99
100      count = limitCount + 1;
101
102     }
103
104
```

```
/*****************************************************************
 * Biblioteca de Controle de Água                                *
 *****************************************************************/


/*****************************************************************/

#include "stm8s.h"
#include "stm8s_io.h"


/*****************************************************************
 * Funções e Procedimentos                                       *
 *****************************************************************/
/*****************************************************************
 * waterControl(enable);                                         *
 *                                                               *
 * Procedimento de controle de água;                             *
 *                                                               *
 * bool -> recebe true para abrir a válvula;                     *
 *                                                               *
 * void -> não retorna valor;                                    *
 *****************************************************************/
void waterControl(bool enable);

/*****************************************************************/
```