```c
//Sistemas Embebidos - Examen parcial 2
//Fernando Cossio Ramirez

#include <avr32/io.h>
#include "compiler.h"

#include <pm.h>
// From module: GPIO - General-Purpose Input/Output
#include <gpio.h>
// From module: Generic board support
#include <board.h>
// From module: INTC - Interrupt Controller
#include <intc.h>
// From module: Interrupt management - UC3 implementation
#include <interrupt.h>

#include <tc.h>

#include <spi.h>

#define PBA_HZ          FOSC0
#define BTN_UP   AVR32_PIN_PB22
#define BTN_DOWN AVR32_PIN_PB23
#define BTN_RIGHT AVR32_PIN_PB24
#define BTN_LEFT AVR32_PIN_PB25
#define BTN_CENTER AVR32_PIN_PB26
#define LED0   AVR32_PIN_PB27
#define LED1   AVR32_PIN_PB28
#define LED2   AVR32_PIN_PA05
#define LED3   AVR32_PIN_PA06

enum btn{NONE, UP, DOWN, LEFT, RIGHT, CENTER};
enum btn btn_pressed = NONE;
uint8_t state = 0;
uint8_t humidity = 0;
uint8_t temperature = 0; //0:NONE, 1:LOW, 2:HIGH
uint8_t timer_configured = 0;
__attribute__ ((__interrupt__));
void btn_interrupt_routine (void);
void leds(uint8_t value);
static void init_tc_output(volatile avr32_tc_t *tc, unsigned int channel);

void state1(void);
void state2(void);
void state3(void);


int main(void){

```

```
50      pm_switch_to_osc0(&AVR32_PM, 16000000, 6);
51      board_init();
52
53      Disable_global_interrupt();
54      INTC_init_interrupts();
55      INTC_register_interrupt(&Botones, 70, 3);
56      INTC_register_interrupt(&Botones, 71, 3);
57
58      uint16_t button_ref [] = {BTN_UP,BTN_DOWN,BTN_RIGHT,BTN_LEFT,BTN_CENTER};
59      for(uint8_t i=0; i<5; i++){
60          gpio_enable_gpio_pin(button_ref[i]);
61          gpio_enable_pin_pull_up(button_ref[i]);
62          gpio_enable
63          _pin_interrupt(button_ref[i],GPIO_FALLING_EDGE);
64      }
65      Enable_global_interrupt();
66      init_tc_output(&AVR32_TC, 2); //Canal 2 como waveform
67      static const gpio_map_t TC_GPIO_MAP =
68      {
69          {86, 2} //GPIO 86, FN especial C, 2
70      };
71      gpio_enable_module(TC_GPIO_MAP, sizeof(TC_GPIO_MAP) / sizeof(TC_GPIO_MAP
          [0]));//Activar Fn especial para TIOA2
72
73      /
          *SPI*************************************************************
          ***/
74      // SPI options.
75      //Mapa SPI
76      static const gpio_map_t SPI_GPIO_MAP = {
77          {AVR32_SPI0_SCK_0_0_PIN , AVR32_SPI0_SCK_0_0_FUNCTION },  // SCK: SPI
            Clock.
78          {AVR32_SPI0_MISO_0_0_PIN, AVR32_SPI0_MISO_0_0_FUNCTION},  // MISO.
79          {AVR32_SPI0_MOSI_0_0_PIN, AVR32_SPI0_MOSI_0_0_FUNCTION},  // MOSI.
80          {AVR32_SPI0_NPCS_3_0_PIN, AVR32_SPI0_NPCS_3_0_FUNCTION}   // NPCS: Chip
              Select
81      };//Pines y funciones SPI
82      spi_options_t spiOptions = {
83          .reg              =  3,   //CHIP SELECT 1
84          .baudrate     =  100000,//BAUDRATE: 100Kbps (sin modulacion)*
85          .bits         =  8,//Número de bits a transmitir: 8
86          .spck_delay   =  48,//Delay antes del SPCK (DLYBS = CLK*DLY = 12M*4u =
            48)*
87          .trans_delay  =  0,//Delay entre transiciones consecutivas (DLYBCT = 0,
              no se especifica un delay)*
88          .stay_act     =  0,//Deselección de perífericos (CSAAT = 0, se
            desactiva en la ultima transferencia)*
89          .spi_mode     =  3,//Modo (CPOL y NCPHA): 0*
90          .modfdis      =  1,//Modo Fault Detection:  1 - Inhabilitado*
```

```
91      };//Estructura SPI
92
93      gpio_enable_module(SPI_GPIO_MAP,
94      sizeof(SPI_GPIO_MAP) / sizeof(SPI_GPIO_MAP[0]));
95      spi_initMaster(AVR32_SPI0_ADDRESS, &spiOptions);
96      // Set SPI selection mode: variable_ps, pcs_decode, delay.
97      spi_selectionMode(AVR32_SPI0_ADDRESS, 0, 0, 0); //PS, PCS_decode, DLYBCS
98      //PS = 0: fija
99      //PCS = 0: sin decodificación
100     //DLYBCS = DLY*CLK, no especificado
101
102     spi_selectChip(AVR32_SPI0_ADDRESS, 3);
103
104     // Enable SPI module.
105     spi_enable(AVR32_SPI0_ADDRESS);
106     spi_setupChipReg(AVR32_SPI0_ADDRESS,&spiOptions,PBA_HZ);
107     /
          **********************************************************************
          ***/
108     while (true)
109     {
110         switch (state) {
111             case 0:
112             //do nothing
113             break;
114             case 1: //programacion de humedad
115             state1();
116             break;
117             case 2://programar temp
118             state2();
119             break;
120             case 3: //contador arriba y abajo
121             state3();
122             break;
123             case 4: //spi
124             state4();
125             break;
126         } //Fin switch
127     } //Fin While
128 }
129
130
131 void state1(void){//increment humidity
132     if(btn_pressed == UP){
133         humidity++;
134         if (humidity > 4) humidity = 1;
135         leds(0b1000>>(humidity-1));
136     }
137     else if(btn_pressed == CENTER){
```

```
138         if(temperature)
139         state = 3;
140         else
141         state = 0;
142         timer_configured = 0;
143     }
144     else if (btn_pressed == RIGHT || btn_pressed == LEFT || btn_pressed ==
          DOWN){
145         humidity = 0; //invalidar seleccion
146     }
147     btn_pressed = NONE;
148 }
149 void state2(void){
150     if(btn_pressed == LEFT){
151         temperature = 1;
152     }
153     else if(btn_pressed == RIGHT){
154         temperature = 2;
155     }
156     else if(btn_pressed == CENTER){
157         if(humidity)
158         state = 3; //pasar a generar PWM
159         else
160         state = 0;
161         timer_configured = 0;
162     }
163     else if (btn_pressed == UP || btn_pressed == DOWN){
164         temperature = 0;
165     }
166     btn_pressed = NONE;
167 }
168 void state3(void){
169     if (!timer_configured){
170         //fPBA=16MHz; fPBA/32=500kHz => TPBA=8e-6 seg
171         //Tpwm= 30ms =>  rc = Tpwm/TPBA = 3750
172         //20%(3750) = 750
173         //40%(3750) = 1500
174         //60%(3750) = 2250
175         //80%(3750) = 3000
176         //Tpwm= 70ms =>  rc = Tpwm/TPBA = 8750
177         //20%(8750) = 1750
178         //40%(8750) = 3500
179         //60%(8750) = 5250
180         //80%(8750) = 7000
181         gpio_set_gpio_pin(86); //Para iniciar PWM en 1
182         if(temperature == 1){
183             //period = 30 ms
184             tc_write_rc(&AVR32_TC0, 2, 3750);
185             tc_write_ra(&AVR32_TC0, 2, 750*humidity);
```

```
186         }
187         else if(temperature ==2){
188             //period = 70ms
189             tc_write_rc(&AVR32_TC0, 2, 8750);
190             tc_write_ra(&AVR32_TC0, 2, 1750*humidity);
191         }
192         tc_start(&AVR32_TC0,2);
193         timer_configured = 1;
194     }
195 
196 }
197 void state4(void){
198     if (humidity && temperature){
199         //send SPI
200         spi_write(AVR32_SPI0_ADDRESS, humidity);
201         spi_write(AVR32_SPI0_ADDRESS, temperature-1);
202         state = 0;
203     }
204 }
205 
206 void leds(uint8_t value){
207     if ((value & 0b1000)>>3)gpio_clr_gpio_pin(LED0); else gpio_set_gpio_pin
        (LED0);
208     if ((value & 0b0100)>>2)gpio_clr_gpio_pin(LED1); else gpio_set_gpio_pin
        (LED1);
209     if ((value & 0b0010)>>1)gpio_clr_gpio_pin(LED2); else gpio_set_gpio_pin
        (LED2);
210     if (value & 0b0001)gpio_clr_gpio_pin(LED3); else gpio_set_gpio_pin(LED3);
211 }//Fin Fn
212 void btn_interrupt_routine (void){
213     if (gpio_get_pin_interrupt_flag(BTN_UP)) {
214         btn_pressed=UP;
215         state=1;
216         gpio_clear_pin_interrupt_flag(BTN_UP);
217     }
218     if (gpio_get_pin_interrupt_flag(BTN_DOWN)){
219         btn_pressed=DOWN;
220         state=4;
221         gpio_clear_pin_interrupt_flag(BTN_DOWN);
222     }
223     if (gpio_get_pin_interrupt_flag(BTN_RIGHT)){
224         btn_pressed=RIGHT;
225         state=2;
226         gpio_clear_pin_interrupt_flag(BTN_RIGHT);
227     }
228     if (gpio_get_pin_interrupt_flag(BTN_LEFT)){
229         btn_pressed=LEFT;
230         state=2;
231         gpio_clear_pin_interrupt_flag(BTN_LEFT);
```

```
232     }
233     if (gpio_get_pin_interrupt_flag(BTN_CENTER)){
234         gpio_clear_pin_interrupt_flag(BTN_CENTER);
235         btn_pressed=CENTER;
236     }
237     if (gpio_get_pin_interrupt_flag(BTN_CENTER)){
238         gpio_clear_pin_interrupt_flag(BTN_CENTER);
239     }
240 } //Fin Botones
241 static void init_tc_output(volatile avr32_tc_t *tc, unsigned int channel){
242     // Options for waveform generation.
243     tc_waveform_opt_t waveform_opt =
244     {
245         .channel  = channel,                                        // Channel
              selection.
246
247         .bswtrg   = 0, //TC_EVT_EFFECT_NOOP,           // Software trigger
              effect on TIOB.
248         .beevt    = 0, //TC_EVT_EFFECT_NOOP,           // External event effect
               on TIOB.
249         .bcpc     = 0, //TC_EVT_EFFECT_NOOP,           // RC compare effect on
              TIOB.
250         .bcpb     = 0, //TC_EVT_EFFECT_NOOP,           // RB compare effect on
              TIOB.
251
252         .aswtrg   = 0, //TC_EVT_EFFECT_NOOP, // Trigger no cambia la salida
253         .aeevt    = 0, //TC_EVT_EFFECT_NOOP, // Trigger no cambia la salida
254         .acpc     = 1, //TC_EVT_EFFECT_SET,               // RC compare effect
              on TIOA.
255         .acpa     = 2, //TC_EVT_EFFECT_CLEAR,          // RA compare effect on
              TIOA.
256
257         .wavsel   = 2, //Simple pendiente, RC determina Periodo, RA Duty
258         .enetrg   = 0, //No hay trigger por evento externo FALSE,
259         .eevt     = 0, //No hay trigger por evento externo
              TC_EXT_EVENT_SEL_TIOB_INPUT,
260         .eevtedg  = 0, //No hay trigger por evento externo TC_SEL_NO_EDGE,
261         .cpcdis   = FALSE, //Se va a generar mas de un perido
262         .cpcstop  = FALSE, //Se va a generar mas de un perido
263
264         .burst    = 0, //Sin Burst, TC_BURST_NOT_GATED
265         .clki     = 0, //Reloj no invertido, TC_CLOCK_RISING_EDGE
266         .tcclks   = 5, // fPBA/128, TC4, TC_CLOCK_SOURCE_TC4
267     };
268
269     // Initialize the timer/counter waveform.
270     tc_init_waveform(tc, &waveform_opt);
271 }
272
```