1. //Sistemas Embebidos - Examen parcial 2
2. //Fernando Cossio Ramirez

3

4 #include <avr32/io.h> 5 #include "compiler.h"

6

1. #include <pm.h>
2. // From module: GPIO - General-Purpose Input/Output
3. #include <gpio.h>
4. // From module: Generic board support
5. #include <board.h>
6. // From module: INTC - Interrupt Controller
7. #include <intc.h>
8. // From module: Interrupt management - UC3 implementation
9. #include <interrupt.h>

16

17 #include <tc.h>

18

19 #include <spi.h>

20

21 #define PBA\_HZ FOSC0 22 #define BTN\_UP AVR32\_PIN\_PB22 23 #define BTN\_DOWN AVR32\_PIN\_PB23 24 #define BTN\_RIGHT AVR32\_PIN\_PB24 25 #define BTN\_LEFT AVR32\_PIN\_PB25 26 #define BTN\_CENTER AVR32\_PIN\_PB26 27 #define LED0 AVR32\_PIN\_PB27 28 #define LED1 AVR32\_PIN\_PB28 29 #define LED2 AVR32\_PIN\_PA05 30 #define LED3 AVR32\_PIN\_PA06

31

# 32 enum btn{NONE, UP, DOWN, LEFT, RIGHT, CENTER};

1. enum btn btn\_pressed = NONE;
2. uint8\_t state = 0; 35 uint8\_t humidity = 0;

36 uint8\_t temperature = 0; //0:NONE, 1:LOW, 2:HIGH

## 37 uint8\_t timer\_configured = 0;

1. \_\_attribute\_\_ ((\_\_interrupt\_\_));
2. void btn\_interrupt\_routine (void);
3. void leds(uint8\_t value);
4. static void init\_tc\_output(volatile avr32\_tc\_t \*tc, unsigned int channel);

42

1. void state1(void);
2. void state2(void);
3. void state3(void);

46

47

48 int main(void){

49

1. pm\_switch\_to\_osc0(&AVR32\_PM, 16000000, 6);
2. board\_init();

52

# 53 Disable\_global\_interrupt();

1. INTC\_init\_interrupts();
2. INTC\_register\_interrupt(&Botones, 70, 3);
3. INTC\_register\_interrupt(&Botones, 71, 3);

57

# 58 uint16\_t button\_ref [] = {BTN\_UP,BTN\_DOWN,BTN\_RIGHT,BTN\_LEFT,BTN\_CENTER};

1. for(uint8\_t i=0; i<5; i++){
2. gpio\_enable\_gpio\_pin(button\_ref[i]);
3. gpio\_enable\_pin\_pull\_up(button\_ref[i]);
4. gpio\_enable
5. \_pin\_interrupt(button\_ref[i],GPIO\_FALLING\_EDGE);
6. }

# 65 Enable\_global\_interrupt();

1. init\_tc\_output(&AVR32\_TC, 2); //Canal 2 como waveform
2. static const gpio\_map\_t TC\_GPIO\_MAP =
3. {
4. {86, 2} //GPIO 86, FN especial C, 2
5. };

## 71 gpio\_enable\_module(TC\_GPIO\_MAP, sizeof(TC\_GPIO\_MAP) / sizeof(TC\_GPIO\_MAP

[0]));//Activar Fn especial para TIOA2

72

1. /

\*SPI\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*/

1. // SPI options.
2. //Mapa SPI
3. static const gpio\_map\_t SPI\_GPIO\_MAP = {
4. {AVR32\_SPI0\_SCK\_0\_0\_PIN , AVR32\_SPI0\_SCK\_0\_0\_FUNCTION }, // SCK: SPI Clock.
5. {AVR32\_SPI0\_MISO\_0\_0\_PIN, AVR32\_SPI0\_MISO\_0\_0\_FUNCTION}, // MISO.
6. {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

1. };//Pines y funciones SPI
2. spi\_options\_t spiOptions = {
3. .reg = 3, //CHIP SELECT 1
4. .baudrate = 100000,//BAUDRATE: 100Kbps (sin modulacion)\*
5. .bits = 8,//Número de bits a transmitir: 8
6. .spck\_delay = 48,//Delay antes del SPCK (DLYBS = CLK\*DLY = 12M\*4u = 48)\*
7. .trans\_delay = 0,//Delay entre transiciones consecutivas (DLYBCT = 0, no se especifica un delay)\*
8. .stay\_act = 0,//Deselección de perífericos (CSAAT = 0, se desactiva en la ultima transferencia)\*
9. .spi\_mode = 3,//Modo (CPOL y NCPHA): 0\*
10. .modfdis = 1,//Modo Fault Detection: 1 - Inhabilitado\*
11. };//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);

1. // Set SPI selection mode: variable\_ps, pcs\_decode, delay.
2. spi\_selectionMode(AVR32\_SPI0\_ADDRESS, 0, 0, 0); //PS, PCS\_decode, DLYBCS
3. //PS = 0: fija
4. //PCS = 0: sin decodificación
5. //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);

1. /

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*/

1. while (true)
2. {
3. switch (state) {
4. case 0:
5. //do nothing
6. break;
7. case 1: //programacion de humedad
8. state1();
9. break;
10. case 2://programar temp
11. state2();
12. break;
13. case 3: //contador arriba y abajo
14. state3();
15. break;
16. case 4: //spi
17. state4();
18. break;
19. } //Fin switch
20. } //Fin While
21. }

129

130

1. void state1(void){//increment humidity
2. if(btn\_pressed == UP){

## 133 humidity++; 134 if (humidity > 4) humidity = 1;

1. leds(0b1000>>(humidity-1));
2. }
3. else if(btn\_pressed == CENTER){
4. if(temperature) 139 state = 3;

140 else

## 141 state = 0; 142 timer\_configured = 0;

1. }
2. else if (btn\_pressed == RIGHT || btn\_pressed == LEFT || btn\_pressed == DOWN){
3. humidity = 0; //invalidar seleccion
4. }
5. btn\_pressed = NONE;
6. }
7. void state2(void){
8. if(btn\_pressed == LEFT){

## 151 temperature = 1;

1. }
2. else if(btn\_pressed == RIGHT){

## 154 temperature = 2;

1. }
2. else if(btn\_pressed == CENTER){

## 157 if(humidity)

1. state = 3; //pasar a generar PWM
2. else

## 160 state = 0; 161 timer\_configured = 0;

1. }
2. else if (btn\_pressed == UP || btn\_pressed == DOWN){

## 164 temperature = 0;

1. }
2. btn\_pressed = NONE;
3. }
4. void state3(void){

## 169 if (!timer\_configured){

1. //fPBA=16MHz; fPBA/32=500kHz => TPBA=8e-6 seg
2. //Tpwm= 30ms => rc = Tpwm/TPBA = 3750
3. //20%(3750) = 750
4. //40%(3750) = 1500
5. //60%(3750) = 2250
6. //80%(3750) = 3000
7. //Tpwm= 70ms => rc = Tpwm/TPBA = 8750
8. //20%(8750) = 1750
9. //40%(8750) = 3500
10. //60%(8750) = 5250
11. //80%(8750) = 7000
12. gpio\_set\_gpio\_pin(86); //Para iniciar PWM en 1

## 182 if(temperature == 1){

1. //period = 30 ms
2. tc\_write\_rc(&AVR32\_TC0, 2, 3750);
3. tc\_write\_ra(&AVR32\_TC0, 2, 750\*humidity);
4. }
5. else if(temperature ==2){
6. //period = 70ms
7. tc\_write\_rc(&AVR32\_TC0, 2, 8750); 190 tc\_write\_ra(&AVR32\_TC0, 2, 1750\*humidity);

191 }

# 192 tc\_start(&AVR32\_TC0,2);

1. timer\_configured = 1;
2. }

195

1. }
2. void state4(void){
3. if (humidity && temperature){
4. //send SPI
5. spi\_write(AVR32\_SPI0\_ADDRESS, humidity);
6. spi\_write(AVR32\_SPI0\_ADDRESS, temperature-1);
7. state = 0;
8. }
9. }

205

1. void leds(uint8\_t value){
2. 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)) {

1. btn\_pressed=UP;
2. state=1;
3. gpio\_clear\_pin\_interrupt\_flag(BTN\_UP);
4. }

## 218 if (gpio\_get\_pin\_interrupt\_flag(BTN\_DOWN)){

1. btn\_pressed=DOWN;
2. state=4;
3. gpio\_clear\_pin\_interrupt\_flag(BTN\_DOWN);
4. }

## 223 if (gpio\_get\_pin\_interrupt\_flag(BTN\_RIGHT)){

1. btn\_pressed=RIGHT;
2. state=2;
3. gpio\_clear\_pin\_interrupt\_flag(BTN\_RIGHT);
4. }

## 228 if (gpio\_get\_pin\_interrupt\_flag(BTN\_LEFT)){

1. btn\_pressed=LEFT;
2. state=2;
3. gpio\_clear\_pin\_interrupt\_flag(BTN\_LEFT);
4. }

## 233 if (gpio\_get\_pin\_interrupt\_flag(BTN\_CENTER)){ 234 gpio\_clear\_pin\_interrupt\_flag(BTN\_CENTER);

1. btn\_pressed=CENTER;
2. }

## 237 if (gpio\_get\_pin\_interrupt\_flag(BTN\_CENTER)){ 238 gpio\_clear\_pin\_interrupt\_flag(BTN\_CENTER);

1. }
2. } //Fin Botones
3. static void init\_tc\_output(volatile avr32\_tc\_t \*tc, unsigned int channel){ 242 // Options for waveform generation.
4. tc\_waveform\_opt\_t waveform\_opt =
5. {
6. .channel = channel, // Channel selection.

246

1. .bswtrg = 0, //TC\_EVT\_EFFECT\_NOOP, // Software trigger effect on TIOB.
2. .beevt = 0, //TC\_EVT\_EFFECT\_NOOP, // External event effect on TIOB.
3. .bcpc = 0, //TC\_EVT\_EFFECT\_NOOP, // RC compare effect on TIOB.
4. .bcpb = 0, //TC\_EVT\_EFFECT\_NOOP, // RB compare effect on TIOB.

251

1. .aswtrg = 0, //TC\_EVT\_EFFECT\_NOOP, // Trigger no cambia la salida
2. .aeevt = 0, //TC\_EVT\_EFFECT\_NOOP, // Trigger no cambia la salida
3. .acpc = 1, //TC\_EVT\_EFFECT\_SET, // RC compare effect on TIOA.
4. .acpa = 2, //TC\_EVT\_EFFECT\_CLEAR, // RA compare effect on TIOA.

256

1. .wavsel = 2, //Simple pendiente, RC determina Periodo, RA Duty
2. .enetrg = 0, //No hay trigger por evento externo FALSE,
3. .eevt = 0, //No hay trigger por evento externo

TC\_EXT\_EVENT\_SEL\_TIOB\_INPUT,

1. .eevtedg = 0, //No hay trigger por evento externo TC\_SEL\_NO\_EDGE,
2. .cpcdis = FALSE, //Se va a generar mas de un perido
3. .cpcstop = FALSE, //Se va a generar mas de un perido

263

1. .burst = 0, //Sin Burst, TC\_BURST\_NOT\_GATED
2. .clki = 0, //Reloj no invertido, TC\_CLOCK\_RISING\_EDGE
3. .tcclks = 5, // fPBA/128, TC4, TC\_CLOCK\_SOURCE\_TC4
4. };

268

269 // Initialize the timer/counter waveform.

## 270 tc\_init\_waveform(tc, &waveform\_opt);

271 }

272