Sistemas Embebidos II

Semestre de Verão de 2010/2011

Terceira actividade prática

1ª Parte - eCos

1 Ambiente de trabalho eCos

Criar directorias de trabalho.

```
se2
      ecos
            app
                  hello
            mylib
            install
            repository
```

Descompactar em repository as fontes eCos disponíveis. Estas fontes são baseadas na versão 3.0 com modificações para adaptação ao LPC-H2106.

Gerar a biblioteca eCos (esta operação requer a assistência do professor).

Verificar a correcta execução do seguinte programa.

main.c

```
#include <stdio.h>

int main(void)
{
  printf("Hello, eCos world!\n");
  return 0;
}
```

Makefile

```
GCC = arm-eabi-gcc
LD = arm-eabi-gcc

BASE_DIR    =../../..
INSTALL_DIR =$(BASE_DIR)/ecos/install/ecos3_install

CFLAGS         = -c -g -Wa,-a=$*.lst -I$(INSTALL_DIR)/include \
                        -I$(BASE_DIR)/ecos/mylib
EXTRACFLAGS    = -Wall -ffunction-sections -fdata-sections

LDFLAGS           = -L$(INSTALL_DIR)/lib -Wl,--gc-sections -Wl,--Map
-Wl,basic.map

VPATH               = $(BASE_DIR)/ecos/mylib

all: main.elf

clean:
      -rm -f *.exe.* main.elf main.hex *.s *.o *.lst *.bak *.map *.ii *.i *.d

%.o: %.c
      $(GCC) -o $*.o $(CFLAGS) $(EXTRACFLAGS) $<
```

```
OBJECTS = main.o

main.elf: $(OBJECTS)
	$(LD) -o $@  $(LDFLAGS) -Ttarget.ld -nostdlib $(OBJECTS)
```

2 Interface HAL para acesso ao hardware.

No programa seguinte apresenta-se um conjunto de funções para actuar um besouro.

Analise o programa e programe a função buzzer_sequence.

```
#include <cyg/kernel/kapi.h>
#include <cyg/hal/hal_io.h>		//  HAL_WRITE...
#include <cyg/hal/hal_diag.h>		//  HAL_DELAY_US
#include "chrono.h"

#define BUZZER_PIN	15

#define LPC2XXX_PINSEL0_GPIO(p)	(~(3 << ((p) << 1)))

void buzzer_init() {
	/* programar a direcção */
	HAL_WRITE_UINT32(CYGARC_HAL_LPC2XXX_REG_IO_BASE +
		CYGARC_HAL_LPC2XXX_REG_IOCLR,
		1 << BUZZER_PIN);
	cyg_uint32 iodir;
	HAL_READ_UINT32(CYGARC_HAL_LPC2XXX_REG_IO_BASE +
		CYGARC_HAL_LPC2XXX_REG_IODIR, iodir);
	HAL_WRITE_UINT32(CYGARC_HAL_LPC2XXX_REG_IO_BASE +
		CYGARC_HAL_LPC2XXX_REG_IODIR,
		iodir |	1 << BUZZER_PIN);

	/* switch pins to GPIO */
	cyg_uint32 pinsel;
	HAL_READ_UINT32(CYGARC_HAL_LPC2XXX_REG_PIN_BASE +
		CYGARC_HAL_LPC2XXX_REG_PINSEL0, pinsel);
	HAL_WRITE_UINT32(CYGARC_HAL_LPC2XXX_REG_PIN_BASE +
		CYGARC_HAL_LPC2XXX_REG_PINSEL0,
		pinsel & LPC2XXX_PINSEL0_GPIO(BUZZER_PIN));
}

void buzzer_on(int on, int frequency) {
	int half_period = 1000000 / frequency / 2;
	cyg_uint32 initial = chrono_start(), duration = msec2tic(on);
//	while (chrono_elapsed(initial) < msec2tic(on)) {
	while (chrono_elapsed(initial) < duration) {
		HAL_WRITE_UINT32(CYGARC_HAL_LPC2XXX_REG_IO_BASE +
			CYGARC_HAL_LPC2XXX_REG_IOSET, 1 << BUZZER_PIN);
		HAL_DELAY_US(half_period)
		HAL_WRITE_UINT32(CYGARC_HAL_LPC2XXX_REG_IO_BASE +
				CYGARC_HAL_LPC2XXX_REG_IOCLR, 1 << BUZZER_PIN);
		HAL_DELAY_US(half_period)
	}
}

void buzzer_off(int off) {
	HAL_WRITE_UINT32(CYGARC_HAL_LPC2XXX_REG_IO_BASE +
				CYGARC_HAL_LPC2XXX_REG_IOCLR, 1 << BUZZER_PIN);
	cyg_thread_delay(msec2tic(off));
}
```

```
void buzzer_sequence(int n, int on, int off, int frequency) {
      ;
}

int main() {
    buzzer_init();
    buzzer_on(4000, 1000);
    buzzer_off(4000);
    buzzer_sequence(6, 100, 500, 1000);
}
```

3 Programação com threads e sincronização.

Reprograme as funções de acesso ao besouro, de modo a manter a mesma interface e semântica de utilização, mas satisfazendo o seguinte requisito: as threads utilizadoras em vez de despender processamento na execução das operações devem entregar pedidos a um thread dedicada ao efeito.

```
void buzzer_init();
void buzzer_sequence(int n, int on, int off, int frequency = 1000);
void buzzer_stop();
void buzzer_start(int frequency = 1000);
void buzzer_beep(int on, int frequency = 1000);
void buzzer_pause(int off);
```

4 Processamento de interrupções.

Verifique o correcto funcionamento do programa seguinte que processa o atendimento de EINT0.

```
#include <cyg/infra/diag.h>
#include <cyg/kernel/kapi.h>
#include <cyg/hal/hal_io.h>

static cyg_interrupt intr_desc;
static cyg_handle_t intr_handle;
static cyg_sem_t ready;

#define CYGNUM_HAL_PRI_HIGH        0

static cyg_uint32 isr(cyg_vector_t vector, cyg_addrword_t data) {
      cyg_interrupt_mask(vector);
      cyg_interrupt_acknowledge(vector);
      return (CYG_ISR_HANDLED | CYG_ISR_CALL_DSR);
}

static void dsr(cyg_vector_t vector, cyg_ucount32 count, cyg_addrword_t data) {
      HAL_WRITE_UINT32(CYGARC_HAL_LPC2XXX_REG_SCB_BASE +
CYGARC_HAL_LPC2XXX_REG_EXTINT,
            CYGARC_HAL_LPC2XXX_REG_EXTxxx_INT0);
      cyg_semaphore_post(&ready);
      cyg_interrupt_unmask(vector);
}

#define THREAD_STACK_SIZE     (8192 / sizeof(int))
static int thread_stack[THREAD_STACK_SIZE];

static cyg_handle_t thread_handle;
static cyg_thread thread_desc;

void thread(cyg_addrword_t index) {
      cyg_uint32 count = 0;
      /* Conectar sinal EINT0 ao pino do circuito integrado */
```

```
      HAL_WRITE_UINT32(CYGARC_HAL_LPC2XXX_REG_PIN_BASE +
CYGARC_HAL_LPC2XXX_REG_PINSEL1, 1);
      cyg_interrupt_unmask(CYGNUM_HAL_INTERRUPT_EINT0);
      while (1) {
            cyg_semaphore_wait(&ready);
            diag_printf("Interrupt %d\n", count++);
      }
}

void cyg_user_start(void) {
      diag_printf("Object Interrupt test\n");

      cyg_semaphore_init(&ready, 0);

      cyg_thread_create(12, thread, 0, "Interrupt test thread",
            &thread_stack, THREAD_STACK_SIZE,
            &thread_handle, &thread_desc);

      cyg_thread_resume(thread_handle);

      cyg_interrupt_create(CYGNUM_HAL_INTERRUPT_EINT0, CYGNUM_HAL_PRI_HIGH, 0,
            isr, dsr, &intr_handle, &intr_desc);

      cyg_interrupt_attach(intr_handle);
    cyg_interrupt_configure(CYGNUM_HAL_INTERRUPT_EINT0, true, true);
    cyg_interrupt_acknowledge(CYGNUM_HAL_INTERRUPT_EINT0);
}
```

O sinal aplicado ao besouro é gerado invertendo uma saída do GPIO a cada meio período. Para uma frequência de 1000 Hz é necessário realizar essa operação de 500 em 500 microsegundos. No programa apresentado acima realizam-se estas temporizações com a função HAL_DELAY_US. Esta função baseia-se no Timer 1 para medir a passagem do tempo, mas consome processamento no teste do valor do Timer **(repository/packages/hal/lpc2xxx/var/v3_0/src/lpc2xxx_misc.c)**.

Propõe-se neste exercício, a fim de aliviar a carga de processamento, executar operação de geração do sinal através de uma interrupção, que ocorre a cada meio ciclo do sinal. (A solução mais adequada seria usar directamente uma saída *match* do Timer.)

Sugere-se, num primeiro passo, que modifique o exemplo anterior para passar a aceitar interrupções do Timer 1. Num segundo passo, integre o processamento desta interrupção no programa realizado no ponto 3.

2ª Parte – Interface Gráfico

1 *Display* gráfico Nokia 6610

A documentação da placa MOD-NOKIA6610 está disponível no *site* da Olimex. http://www.olimex.com/dev/mod-nokia6610.html. Os *displays* disponíveis incorporam o controlador EPSON S1D15G00.

Desenhe um esquema de ligações da placa MOD-NOKIA6610 ao LPC2106. Além da alimentação e dos sinais de comunicação SPI, ligue também o sinal de *reset* (LCD_RST) e o de controlo do *backlight* (LCD_BL).

Proceda às ligações e comece por experimentar o controlo de *backlight*.

```
void lcd_init() {

      /* Seleccionar os pinos de RESET E BACKLIGHT para o GPIO */
      U32 pinsel = io_read_u32(LPC210X_SCB + LPC2XXX_PINSEL0);
      io_write_u32(LPC210X_SCB + LPC2XXX_PINSEL0,
            pinsel & ~(3 << BACKLIGHT_PIN * 2) & ~(3 << RESET_PIN * 2));

      /* desactivar o sinal reset e backlight */
      io_write_u32(LPC210X_GPIO + LPC2XXX_IOSET,
            1 << BACKLIGHT_PIN | 1 << RESET_PIN);
      /* e programá-los como saídas */
      U32 iodir = io_read_u32(LPC210X_GPIO + LPC2XXX_IODIR);
      io_write_u32(LPC210X_GPIO + LPC2XXX_IODIR,
            iodir | 1 << BACKLIGHT_PIN | 1 << RESET_PIN);

      /* executar reset */
      io_write_u32(LPC210X_GPIO + LPC2XXX_IOCLR, 1 << RESET_PIN);
      io_write_u32(LPC210X_GPIO + LPC2XXX_IOSET, 1 << RESET_PIN);
}

void lcd_backlight(int on) {
      if (on)
            io_write_u32(LPC210X_GPIO + LPC2XXX_IOSET, 1 << BACKLIGHT_PIN);
      else
            io_write_u32(LPC210X_GPIO + LPC2XXX_IOCLR, 1 << BACKLIGHT_PIN);
}
```

A comunicação com o controlador S1D15G00 faz-se por uma interface série compatível com SPI, em modo 0, com palavras de 9 bits. No controlador SPI, integrado no microcontrolador LPC2106, na versão 00, o número de bits de dados é fixo em 8. A solução viável para comunicar com o controlador é usar a implementação SPI desenvolvida na primeira actividade prática.

```
Spi_device spis[] = {
      {
            8,    /* GPIO chip select position */
            10,   /* o sinal SCLK terá a frequência = PCLK / este valor */
            0,    /* modo SPI = 0;  cpol = 0, cpha = 0*/
            9     /* number of bits per word */
      }
};
```

As comunicações com o controlador são organizadas em sequências formadas por uma palavra de comando seguida de zero ou mais palavras de dados. O bits de maior peso define o tipo de palavra: a um indica palavra de dados e a zero indica palavra de comando. Os restantes 8 bits identificam o comando ou transportam os parâmetros do comando.

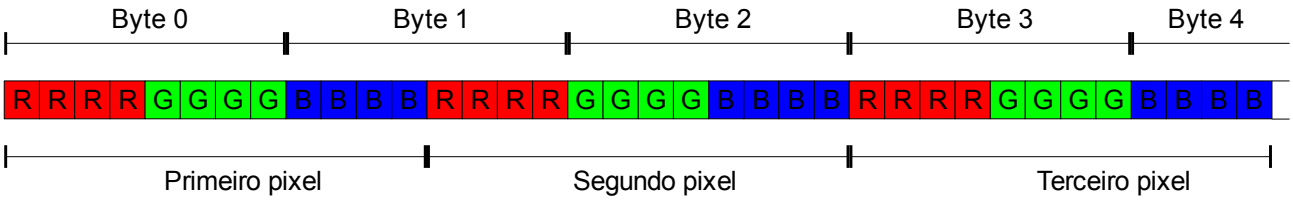
```
static inline void write_command(U8 command) {
      spi_write(spi, ~0x100 & command);
}

static inline void write_data(U8 data) {
      spi_write(spi, 0x100 | data);
}
```

Para iniciar o *display* aplica-se a sequência de comandos recomendada na página 54 do manual.

```
void lcd_init() {

      ...

      spi_transaction_begin(spi);

      write_command(DISCTL);  /*    display control */
      write_data(0x00);       /* P1: 0x00 = 2 divisions,
                                    switching period=8 (default) */
      write_data(32);         /* P2: nlines/4 - 1 = 132/4 - 1 = 32) */
      write_data(00);         /* P3: 0x00 = no inversely highlighted lines */
      write_command(COMSCN);  /* COM scan */
      write_data(1);          /* P1: 0x01 = Scan 1->80, 160<-81 */
      write_command(OSCON);   /* Internal oscilator ON */
      write_command(SLPOUT);  /* sleep out */
      write_command(VOLCTR);  /* Voltage control (contrast setting) */
      write_data(contrast);   /* P1 = 32 volume value (experiment with this
                                    value to get the best contrast) */
      write_data(3);          /* P2 = 3 resistance ratio
                                    (only value that works) */
      write_command(PWRCTR);  /* Power control */
      write_data(0x0f);       /* reference voltage regulator on,
                                    circuit voltage follower on, BOOST ON */
      write_command(DATCTL);  /* Data control */
      write_data(0x00);       /* P1: 0x00 = page address normal,
                                    column address normal,
                                    address scan in column direction */
      write_data(0x00);       /* P2: 0x00 = RGB sequence (default value) */
      write_data(0x02);       /* P3: 0x02 = Grayscale -> 16
                                    (selects 12-bit color, type A) */
      write_command(DISON);   /* Display on */

      spi_transaction_end(spi);
}
```

Os comandos CASET e PASET do controlador permitem o endereçamento à memória de vídeo através das coordenadas gráficas. Com estes comandos é possível definir uma área de intervenção rectangular em qualquer posição e com qualquer dimensão.
Depois de definida a zona de memória a intervir descarregam-se, sequencialmente, os dados necessários. O controlador ajusta automaticamente os endereços da memória de vídeo, à medida que os dados são transferidos.
Todas as funções básicas de desenho como, o desenho de pixel, o desenho de linhas horizontais e verticais e o preenchimento de áreas rectangulares são baseadas nos comandos CASET e PASET.

Na memória de vídeo cada componente de cor ocupa 4 bits, o que corresponde 1,5 byte por pixel. A transferência de dados para a memória é feita bytes-a-byte em que cada byte transporta duas componentes de cor. As transferências devem ser feitas em grupos de três bytes. O primeiro byte transporta as componentes R e G do primeiro pixel, o segundo byte a componentes B do primeiro pixel e a componente R do segundo pixel e o terceiro byte as componentes G e B do segundo pixel.
Mesmo nas situações em que o número de pixeis é ímpar é necessário escrever um número de bytes múltiplo de três.




```
void lcd_draw_point(int x, int y, int color) {
      spi_select(spi);
      write_command(CASET);         /* Column address set (command 0x2A) */
      write_data(x);
      write_data(x);
      write_command(PASET);         /* Page address set (command 0x2B) */
      write_data(y);
      write_data(y);
      write_command(RAMWR);         /* write memory */
      write_data(color >> 4);
      write_data((color & 0xf) << 4);
      write_data(0);
      write_command(NOP);
      spi_unselect(spi);
}

void lcd_draw_hor_line(int x, int y, int  dx, int color) {
      int i;
      spi_select(spi);
      write_command(CASET);
      write_data(x);
      write_data(x + dx - 1);
      write_command(PASET);
      write_data(y);
      write_data(y);
      write_command(RAMWR);
      for (i = (dx + 1) / 2; i > 0; --i) {
            write_data(color >> 4);
            write_data(((color << 4) & 0xf0) | ((color >> 8) & 0xf) );
            write_data(color);
      }
      write_command(NOP);
      spi_unselect(spi);
}

void lcd_draw_ver_line(int x, int y, int  dy, int color) {
      int i;
      spi_select(spi);
      write_command(CASET);
      write_data(x);
      write_data(x);
      write_command(PASET);
      write_data(y);
      write_data(y + dy - 1);
      write_command(RAMWR);
      for (i = (dy + 1) / 2; i > 0; --i) {
            write_data(color >> 4);
            write_data(((color << 4) & 0xf0) | ((color >> 8) & 0xf) );
            write_data(color);
      }
      write_command(NOP);
      spi_unselect(spi);
```

```
}

void lcd_fill_rectangle(int x, int y, int dx, int dy, int color) {
      int i;
      spi_select(spi);
      write_command(CASET);
      write_data(x);
      write_data(x + dx - 1);
      write_command(PASET);
      write_data(y);
      write_data(y + dy - 1);
      write_command(RAMWR);
      for (i = (dx * dy + 1) / 2 + 1; i > 0; --i) {
            write_data(color >> 4);
            write_data(((color << 4) & 0xf0) | ((color >> 8) & 0xf) );
            write_data(color);
      }
      write_command(NOP);
      spi_unselect(spi);
}

void lcd_copy_rectangle(int x, int y, int dx, int dy, U8 * bitmap) {
      int i;
      spi_select(spi);
      write_command(CASET);
      write_data(x);
      write_data(x + dx - 1);
      write_command(PASET);
      write_data(y);
      write_data(y + dy - 1);
      write_command(RAMWR);
      for (i = (dx * dy + 1) / 2 + 1; i > 0; --i) {
            write_data(*bitmap++);
      }
      write_command(NOP);
      spi_unselect(spi);
}
```

Faça um programa de teste para desenhar linhas verticais e horizontais e uns rectângulos com as áreas coloridas de diferentes cores.

Integre o programa anterior com o eCos. Basicamente, terá que substituir as funções de acesso aos periféricos pelas funções HAL_WRITE_UINT32 e HAL_READ_UINT32 e os identificadores dos registos e respectivos valores.

2 Aplicação com interface gráfica de utilizador

Em anexo, é fornecida uma aplicação que utiliza uma interface gráfica de utilizador, suportada sobre a interface de display desenvolvida no ponto anterior. Esta aplicação executa a função de um relógio despertador.

Código fornecido:

widget.c widget.h – objectos gráficos.
rtc.c rtc.h – funções auxiliares para manipulação do relógio.
button.c button.h – funções de acesso ao teclado.
main.c – programa principal.

Esta aplicação está estruturada segundo um modelo de máquina de estados em que a maior parte do processamento é consumido na pesquisa de eventos – teclas premidas, avanço do tempo, etc.
Sugere-se neste exercício, a substituição deste processamento de pesquisa de eventos, por uma fila de eventos onde a aplicação aguarda, em espera passiva, a chegada de um novo evento.
Estruture o software de modo que os eventos sejam gerados de forma independente. Por exemplo, os eventos do teclado serão gerados pelo módulo de teclado, os do relógio pelo módulo do relógio, etc. De tal modo que, acrescentar ou remover módulos de geração de eventos, exija as mínimas alterações no software existente.