UNIVERSIDAD DE ALCALÁ ESCUELA POLITÉCNICA SUPERIOR DEPARTAMENTO DE ELECTRÓNICA		GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN	
ASIGNATURA	SISTEMAS ELECTRÓNICOS DIGITALES AVANZADOS	FECHA	NOVIEMBRE 2015
APELLIDOS, NOMBRE	SOLUCIÓN	GRUPO	

PRUEBA DE EVALUACIÓN INTERMEDIA

Ejercicio 1

Se ha diseñado una aplicación cuyo código fuente se muestra en el **Anexo 1**. Se han programado tres interrupciones externas; EINT0, EINT1 y EINT2; así como el timer asociado al Cortex-M3, SysTick. Como se puede observar en el código fuente del proyecto facilitado, una vez que se entra en una rutina de tratamiento de interrupción EINTx, se permanece en ella hasta que el GPIO asociado (P2.0 para EINT0, P2.1 para EINT1 y P2.2 para EINT2) pase a nivel bajo (L).

Se recuerda que los GPIO, tras el RESET, están configurados como INPUTs y con un resistor de pull-up interno, así que, por defecto sus entradas están a nivel H.

Las prioridades de las interrupciones programadas, de máxima prioridad a mínima prioridad, son las siguientes:

Fuente	Nivel de Prioridad
SysTick	1
EINT2	2
EINT1	3
EINT0	4

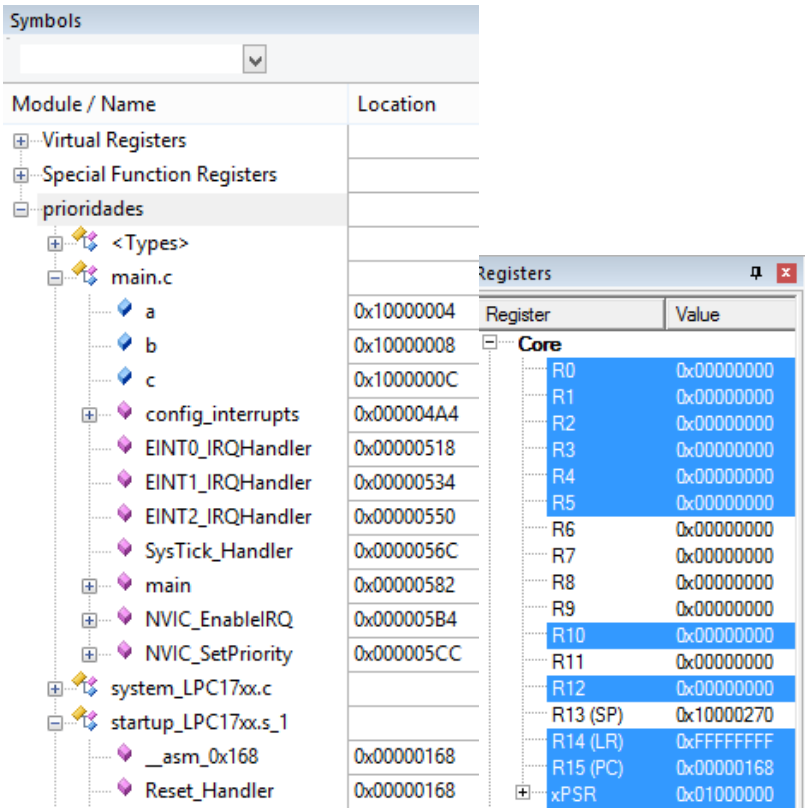
El SysTick se ha configurado, usando la correspondiente función CMSIS, para interrumpir al Cortex-M3 periódicamente (ojo con la frecuencia de CPU programada).

El modelo de excepciones del Cortex-M3 se ha programado con el **grupo de prioridad 4**.

a) Atendiendo al contenido del fichero *startup_LPC17xx.s*, de los registros internos, y de la ventana *Symbols* que se muestran en la siguiente figura, inicializa los vectores de interrupción de la aplicación.

```
DCD     __initial_sp              ; Top of Stack
DCD     Reset_Handler             ; Reset Handler
DCD     NMI_Handler               ; NMI Handler
DCD     HardFault_Handler         ; Hard Fault Handler
DCD     MemManage_Handler         ; MPU Fault Handler
DCD     BusFault_Handler          ; Bus Fault Handler
DCD     UsageFault_Handler        ; Usage Fault Handler
DCD     0                         ; Reserved
DCD     0                         ; Reserved
DCD     0                         ; Reserved
DCD     0                         ; Reserved
DCD     SVC_Handler               ; SVCall Handler
DCD     DebugMon_Handler          ; Debug Monitor Handler
DCD     0                         ; Reserved
DCD     PendSV_Handler            ; PendSV Handler
DCD     SysTick_Handler           ; SysTick Handler

; External Interrupts
DCD     WDT_IRQHandler            ; 16: Watchdog Timer
DCD     TIMER0_IRQHandler         ; 17: Timer0
DCD     TIMER1_IRQHandler         ; 18: Timer1
DCD     TIMER2_IRQHandler         ; 19: Timer2
DCD     TIMER3_IRQHandler         ; 20: Timer3
DCD     UART0_IRQHandler          ; 21: UART0
DCD     UART1_IRQHandler          ; 22: UART1
DCD     UART2_IRQHandler          ; 23: UART2
DCD     UART3_IRQHandler          ; 24: UART3
DCD     PWM1_IRQHandler           ; 25: PWM1
DCD     I2C0_IRQHandler           ; 26: I2C0
DCD     I2C1_IRQHandler           ; 27: I2C1
DCD     I2C2_IRQHandler           ; 28: I2C2
DCD     SPI_IRQHandler            ; 29: SPI
DCD     SSP0_IRQHandler           ; 30: SSP0
DCD     SSP1_IRQHandler           ; 31: SSP1
DCD     PLL0_IRQHandler           ; 32: PLL0 Lock (Main PLL)
DCD     RTC_IRQHandler            ; 33: Real Time Clock
DCD     EINT0_IRQHandler          ; 34: External Interrupt 0
DCD     EINT1_IRQHandler          ; 35: External Interrupt 1
DCD     EINT2_IRQHandler          ; 36: External Interrupt 2
DCD     EINT3_IRQHandler          ; 37: External Interrupt 3
DCD     ADC_IRQHandler            ; 38: A/D Converter
```



Reset

DIR	DATO
0x0000.0000	0x70
0x0000.0001	0x02
0x0000.0002	0x00
0x0000.0003	0x10
0x0000.0004	0x69
0x0000.0005	0x01
0x0000.0006	0x00
0x0000.0007	0x00

EXT 0

DIR	DATO
0x0000.0088	0x19
0x0000.0089	0x05
0x0000.008A	0x00
0x0000.008B	0x00

EXT 1

DIR	DATO
0x0000.008C	0x35
0x0000.008	0x05
0x0000.008E	0x00
0x0000.008F	0x00

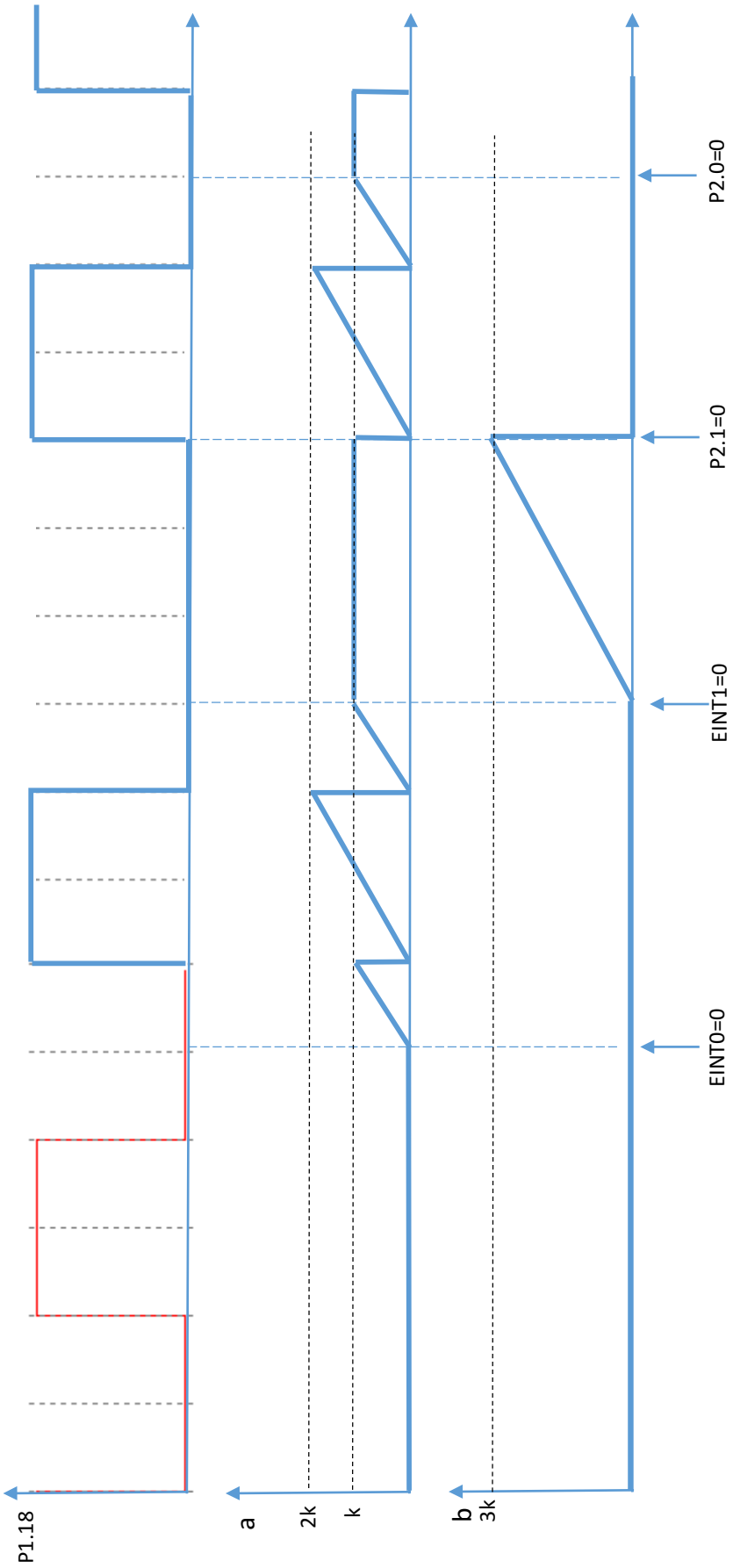
EXT 2

DIR	DATO
0x0000.0090	0x51
0x0000.0091	0x05
0x0000.0092	0x00
0x0000.0093	0x00

SysTick

DIR	DATO
0x0000.003C	0x6D
0x0000.003D	0x05
0x0000.003E	0x00
0x0000.003F	0x00

b) Complete la evolución temporal de las variables a, b y el estado del pin P1.18 en el diagrama temporal que se muestra a continuación.



Ejercicio 2

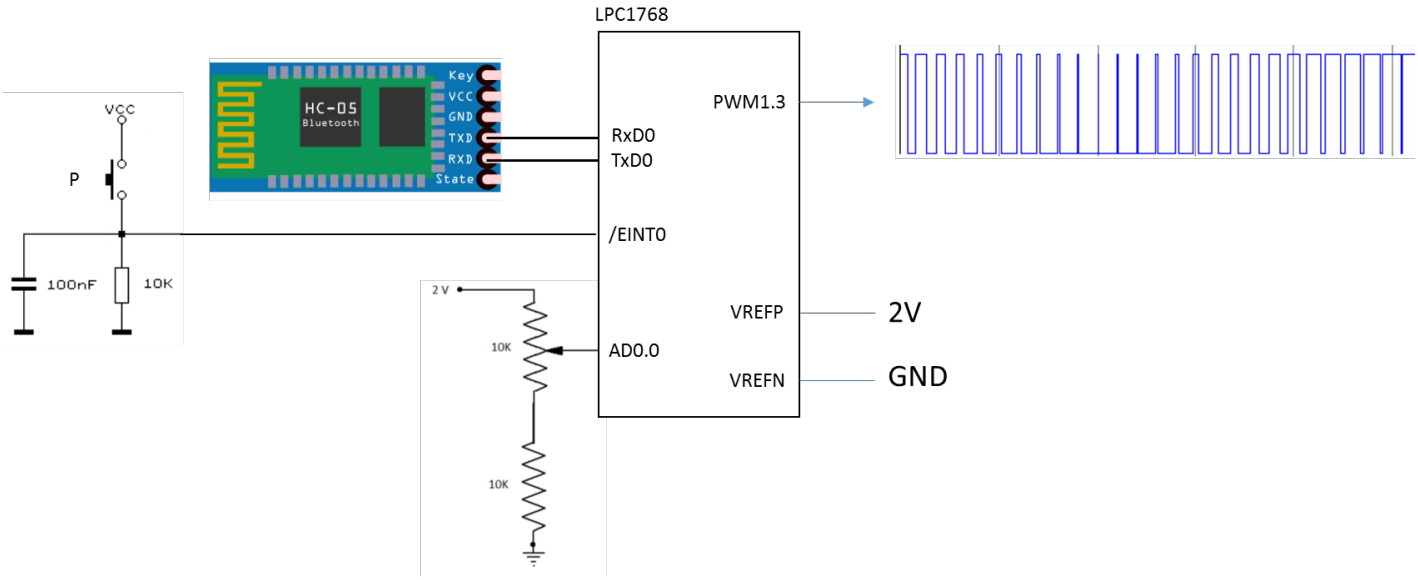
Se pretende diseñar un modulador de señal PWM cuyo ciclo de trabajo varíe (0-100%) al actuar sobre un potenciómetro, y cuya frecuencia pueda seleccionarse entre 4 valores (**1kHz - 2 kHz – 5kHz** y **10kHz**) al actuar sobre un pulsador **P** o a través de la interfaz serie asíncrona.

Condiciones de diseño:

1. Considere la salida **PWM1.3**.
2. El potenciómetro se conectará a la entrada analógica AD0.0 (canal 0). Su posición entre un extremo y otro hará que la tensión varíe entre **1V y 2V** modificando el ciclo de trabajo entre el 0 y 100%. Modifique si cree necesario las entradas **Vrefp** y **Vrefn** del ADC. Configure el ADC para muestrear el **canal 0** a una frecuencia de **1kHz** a través del **Timer 1** (ver opciones de inicio de conversión). Configure **el reloj del ADC** a la mínima frecuencia posible.
3. Utilice la interrupción del **Timer 0** para leer periódicamente cada **50ms** el registro ADDR0 (Así no es necesaria la interrupción del ADC) modificando el ciclo de trabajo de la señal PWM sólo si existe un cambio en su valor.
4. El **pulsador P** que permite seleccionar la frecuencia de la señal PWM se conectará a la **/EINT0.**
5. Cada vez que ocurre un cambio en la frecuencia de la señal de salida se ha de enviar por *Bluetooth* el valor de dicha frecuencia en ASCII. El puerto serie (**UART0**) se configurará a 9600 baudios, 8 bits/dato, 1 bit de Stop y sin paridad. Tanto la transmisión como la recepción se hará por **interrupción**.
6. El programa principal quedará a la espera de interrupciones, hasta completar la recepción por Bluetooth de la frecuencia deseada de la señal PWM en Hz, a través de la interfaz serie.
   NOTA: El Anexo 2 muestra el código de las funciones principales de la UART.

a) Dibuja el diagrama de bloques de interconexión del LPC1768 en el sistema, donde se aprecie cláramente los pines a los que se conectan los dispositivos.

pastor



b) Completa el código fuente y los comentarios de la aplicación

```
#include <LPC17xx.H>
#include <stdlib.h>
#include <stdio.h>
#include "uart.h"
#define F_pclk 25e6
char cadena[10];      // Para convertir con sprintf
char buffer[30];      // Buffer de recepción de 30 caracteres
char *ptr_rx;         // puntero de recepción
char rx_completa;     // Flag de recepción de cadena (se activa con 0x0D ó return)
char *ptr_tx;         // puntero de transmisión
char tx_completa;     // Flag de transmisión de cadena(al enviar el carácter null)
char opcion;
uint32_t F_pwm=1000;  // 1000 Hz valor inicial
uint32_t dato_ADC;
float ciclo;

void config_pwm3(void)
{
LPC_SC->PCONP|=(1<<6);         // Power ON módulo PWM
LPC_PINCON->PINSEL3|=(2<<10); // P1.21 salida PWM (PWM1.3)
LPC_PWM1->MR0 = (F_pclk/F_pwm)-1;
ciclo=(float)(dato_ADC-2047)/2048; // entre 1 y 2V
LPC_PWM1->MR3=(uint32_t)LPC_PWM1->MR0*ciclo; // Actualizo el ciclo de trabajo
LPC_PWM1->LER|=(1<<3)|(1<<0);
LPC_PWM1->PCR|=(1<<11); //configurado el ENA3 (PWM1.3)
LPC_PWM1->MCR|=(1<<1);
LPC_PWM1->TCR|=(1<<0)|(1<<3);
}

/* Timer 0 en modo Match (reset T0TC on Match 0)*/
/* Genera interrp. periódicas cada 50ms*/
void init_TIMER0(void)
{
LPC_SC->PCONP|=(1<<1);      // Power Timer 0
LPC_TIM0->MCR = 0x03 ;      // Reset TC on Match, and Interrupt!
LPC_TIM0->MR0 = (F_pclk*50e-3)-1; //
LPC_TIM0->EMR = 0x02;       //
LPC_TIM0->TCR = 0x01;       // Start Timer
NVIC_SetPriority(TIMER0_IRQn,0); // Nivel 0
NVIC_EnableIRQ(TIMER0_IRQn);// Hab. Interrup. Timer 0
}

// Timer 0 interrumpe periódicamente a 20 Hz
// Leemos ADDR0 y modificamos el ciclo de trabajo si hay cambio
void TIMER0_IRQHandler(void)
{
static uint32_t temp;
LPC_TIM0->IR|= (1<<0);           // Borrar flag
dato_ADC= LPC_ADC->ADDR0>>4)&0xFFF;
if(temp!=  dato_ADC) {          // Cambio en la entrada del ADC
      ciclo=(float)(dato_ADC-2047)/2048;
      LPC_PWM1->MR3=(uint32_t)LPC_PWM1->MR0*ciclo;
      LPC_PWM1->LER|=(1<<3);
      temp= LPC_ADC->ADDR0>>4)&0xFFF; // Almaceno el nuevo valor
      }
}
```

```
void init_ADC(void)
{
LPC_SC->PCONP|= (1<<12);         // Power ON
LPC_PINCON->PINSEL1|= (1<<14);  // ADC input= P0.23 (AD0.0)
LPC_PINCON->PINMODE1|= (2<<14); // Deshabilita pullup/pulldown
LPC_ADC->ADCR= (0x01<<0)|  // Canal 0
               (0xFF<<8)|  // CLKDIV=255; Fclk_ADC=25Mhz/256= 97.656 kHz
               (0x01<<21)| // PDN=1
               (0x06<<24); // Inicio de conversión por MAT1.0
}

//Timer 1 en modo Output Compare (reset T1TC on Match 0)
//Cada 2 Match se provoca el INICIO DE CONVERSIÓN DEL ADC */

void init_TIMER1(void)
{
LPC_SC->PCONP|=(2<<1);                 // Power ON T1
LPC_TIM1->MCR = 0x02;                  // Reset on Match 0
LPC_TIM1->MR0 = (F_pclk/F_muestreo/2)-1; //DOS Match for Start Conversion
LPC_TIM1->EMR = (3<<4);                // Toggle salida
LPC_TIM1->TCR = 0x01;                  // Start T1
}


void config_EINT0(void)
{
LPC_PINCON->PINSEL4|=(0x01<<20);       // P2.10 es entrada interrup. EXT 0
LPC_SC->EXTMODE|=(1<<0);               // Por Flanco,
LPC_SC->EXTPOLAR=0;                    // de bajada
NVIC_SetPriority(EINT0_IRQn, 2);       // Nivel 2
NVIC_EnableIRQ(EINT0_IRQn);            // Hab. Interrup. Externa 0
}


void EINT0_IRQHandler()
{
LPC_SC->EXTINT=(1<<0); // Borrar flag Externa 0

opcion++;
      opcion&=0x03;
      switch(opcion){
            case 0:
            F_pwm=1000;
            break;

            case 1:
            F_pwm=2000;
            break;

            case 2:
            F_pwm=5000;
            break;

            case 3:
            F_pwm=10000;
            break;
      }
```

```
LPC_PWM1->MR0 = (F_pclk/F_pwm)-1;

dato_ADC=(LPC_ADC->ADDR0>>4)&0xFFF;
ciclo=(float)(dato_ADC-2047)/2048;

LPC_PWM1->MR3= (uint32_t)LPC_PWM1->MR0*ciclo;
LPC_PWM1->LER|=(1<<3)|(1<<0);
LPC_PWM1->TC = 0;

sprintf(cadena,"%d\n\r",F_pwm);
tx_cadena_UART0(cadena);

}


int main(void)
{

ptr_rx=buffer;                    // inicializa el puntero de recepción
NVIC_SetPriorityGrouping(2);      // 32 niveles de prioridad preemptive
init_TIMER1();
init_ADC();
config_EINT0();
uart0_init(9600);
init_TIMER0();
config_pwm3();


tx_cadena_UART0("Introduce la frecuencia en Hz:\n\r");

while(1){

      if(rx_completa){


                  rx_completa=0;
                  F_pwm=atoi(buffer);       // Convertimos a entero
                  LPC_PWM1->MR0 = (F_pclk/F_pwm)-1;
                  ciclo=(float)(dato_ADC-2047)/2048;
                  LPC_PWM1->MR3=(uint32_t)LPC_PWM1->MR0*ciclo;
                  LPC_PWM1->LER|=(1<<3)|(1<<0);
                  LPC_PWM1->TC = 0;
                  tx_cadena_UART0("Introduce la frecuencia en Hz:\n\r");

      }

   }
}
```

Ejercicio 3

La Figura 1 muestra el control de una máquina de "vending". En la Figura 2 se muestra la lista los eventos del sistema de control junto con su significado.

a) Describe las transiciones que se producen desde que un usuario inserta una moneda.

b) El control de de la máquina de "vending" tiene un "bug" que hace que los usuarios puedan hacer trampas. Encuentra el fallo.

c) Indica cómo modificar el control para evitar el fallo.

d) En la Figura 3 se añaden dos nuevos eventos para distinguir monedas de 50c y de 1 €. Modifica el Statechart para que se sirvan bebidas cuando se haya introducido 1€. El sistema no devuelve cambio.

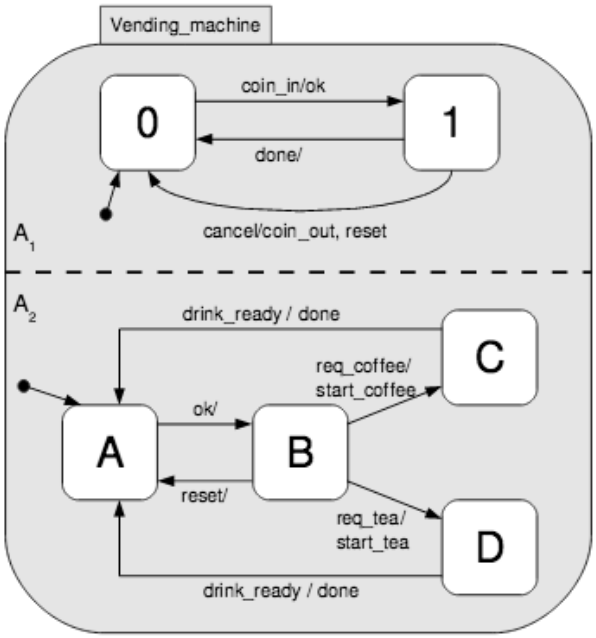


Figura 1: Modelo de comportamiento del sistema de control

Evento	Generado por	Consumido por	Significado
COIN_IN	Entorno	A_1	El usuario inserta moneda
CANCEL	Entorno	A_1	El usuario presiona el botón 'Cancelar'
REQ_COFEE	Entorno	A_2	El usuario presiona el botón ''Café'
REQ_TEA	Entorno	A_2	El usuario presiona el botón 'Té'
DRINK_READY	Entorno	A_2	La bebida está lista
COIN_OUT	A_1	Entrono	La moneda retorna al usuario
START_COFFEE	A_2	Entrono	Inicia la preparación del café
START_TEA	A_2	Entrono	Inicia la preparación del té
OK	A_1	A_2	Suficientes monedas insertadas
RESET	A_1	A_2	Devolver monedas al usuario
DONE	A_2	A_1	Bebida servida

Figura 2: Tabla de eventos del sistema de control

Evento	Generado por	Consumido por	Significado
COIN_50C_IN	Entorno	A_1	El usuario inserta moneda de 50c
COIN_1E_IN	Entorno	A_1	El usuario inserta moneda de 1€

Figura 3: Tabla de eventos añadidos al sistema de control

a) Antes de que el usuario realice ninguna acción, el sistema está en el estado [A1-0, A2-2]. Cuando el Entorno detecta que se ha introducido moneda, se activa el evento COIN_IN que hace que el sistema pase al estado [A1-1, A2-A] señalizando mediante OK que hay suficientes monedas insertadas, provocando el cambio de estado a [A1-1, A2-B]. En este punto, el sistema queda esperando a que el usuario seleccione la bebida.

Cuando se selecciona la bebida el Entrono genera los eventos REQ_COFFE o REQ_TEA que provocan el cambio al estado [A1-1, A2-C] o [A1-1, A2-D] respectivamente.

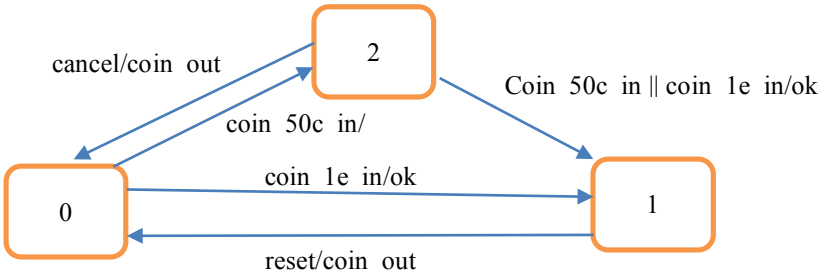
Cuando el Entorno finaliza la preparación de la bebida, lo señaliza con DRINK_READY lo que provoca un cambio al estado [A1-1, A2-A] y genera la señal DONE que hace que el sistema vuelva al estado inicial [A1-0, A2-2].

Si mientras el sistema está funcionando, el usuario intenta cancelar el proceso, el Entorno genera el evento CANCEL que provocaría un cambio del estado A1-1 al A1-0, se generaría la señal COIN_OUT, devolviendo el dinero al usuario y volvería al estado A2-A por la activación de la señal RESET si se encontrara en A2-B.

b) Si el usuario intenta cancelar la operación mientras se está preparando la bebida (estando en [A1-1, A2-C] o en [A1-1, A2-D]), se devuelve el dinero pero sigue preparando la bebida entregando al usuario el dinero y la bebida.

c) La mejor forma de evitarlo es que sólo se pueda cancelar el proceso antes de seleccionar la bebida, es decir cuando el sistema está en [A1-1, A2-B]. Esto se puede conseguir asociando el evento CANCEL a la transición de A1-B a A1-A e indicando que active en ese momento la señal RESET. La transición de A1-1 a A1-0 en este caso se activaría con RESET y como salida produciría COIN_OUT.

d) Se añade un estado A1-2 en el que debería estar si la primera moneda que se introduce es de 50c. En caso de que el usuario intente cancelar, se debería devolver el dinero cuando esté en el estado A1-2. Esto se hace con la señal CANCEL directamente ya que RESET no se activa hasta que no está en el estado A2-B, es decir, hasta que no se envía la señal OK.



Anexo 1. Código del programa

```
#include <LPC17xx.H>
#define Ftick 1              // Frecuencia (Hz) de interrupción del SYSTICK (ojo con la frecuencia de CPU)
volatile uint32_t a,b,c;     // variables auxiliares

//Inicializa las interrupciones
void config_interrupts()
{
       LPC_PINCON->PINSEL4 |= 0x01<<20;             // P2.10 es entrada interrup. EXT 0
       LPC_PINCON->PINSEL4 |= 0x01<<22;             // P2.11 es entrada interrup. EXT 1
       LPC_PINCON->PINSEL4 |= 0x01<<24;             // P2.12 es entrada interrup. EXT 2
       LPC_SC->EXTMODE       |= 1<<2 | 1<<1 | 1<<0; // EINT1,EINT2,EINT3 activas por Flanco
       LPC_SC->EXTPOLAR=0;                          // Por flanco de bajada

       // Asignación de prioridades
       NVIC_SetPriorityGrouping(4);
       NVIC_SetPriority(EINT0_IRQn, 4);            //
       NVIC_SetPriority(EINT1_IRQn, 3);            //
       NVIC_SetPriority(EINT2_IRQn, 2);            //
       NVIC_SetPriority(SysTick_IRQn, 1);          //

       // Habilitación de las interrupciones
       NVIC_EnableIRQ(EINT0_IRQn); // sin CMSIS: NVIC->ISER[0]=(1<<18);
       NVIC_EnableIRQ(EINT1_IRQn); // sin CMSIS: NVIC->ISER[0]=(1<<19);
       NVIC_EnableIRQ(EINT2_IRQn); // sin CMSIS: NVIC->ISER[0]=(1<<20);
}

//-----------------  ISR EINT0  ------------------------------------------------------
void EINT0_IRQHandler()
{
       LPC_SC->EXTINT = 1<<0;                       // Borrar flag Externa 0
       while(LPC_GPIO2->FIOPIN&(1<<0)) a++;         // No sale de la ISR_EINT0 mientras P2.0 = H
}

//-----------------  ISR EINT1  ------------------------------------------------------
void EINT1_IRQHandler()
{
       LPC_SC->EXTINT = 1<<1;                       // Borrar flag  Externa 1
       while(LPC_GPIO2->FIOPIN&(1<<1)) b++;         // No sale de la ISR_EINT1 mientras P2.1 = H
}


//-----------------  ISR EINT2  ------------------------------------------------------
void EINT2_IRQHandler()
{
       LPC_SC->EXTINT = 1<<2;                       // Borrar flag Externa 2
       while(LPC_GPIO2->FIOPIN&(1<<2)) c++;         // No sale de la ISR_EINT2 mientras P2.2 = H
}


//-----------------  ISR SysTick   ---------------------------------------------------
void SysTick_Handler(void)                  // Se ejecuta periódicamente a Ftick (Hz)
{
       LPC_GPIO1->FIOPIN ^= 1<<18;          // Conmuta P1.18
       a=0;
       b=0;
       c=0;
}


//-----------------   main  ----------------------------------------------------------
int main ()
{
  LPC_GPIO1->FIODIR|=(1<<18);               // P1.18 salida
  SysTick_Config(SystemCoreClock/Ftick); // Valor de RELOAD (N° de cuentas  del SYSTICK hasta llegar a cero)
  config_interrupts();
  while(1);                                 // JAMAS se retorna de main()
}
```

Anexo 2. Funciones UART

```
#include <LPC17xx.h>
#include "uart.h"


/*
 * UART0 interrupt handler
 */

void UART0_IRQHandler(void) {

    switch(LPC_UART0->IIR&0x0E) {

        case 0x04:                                  /* RBR, Receiver Buffer Ready */
        *ptr_rx=LPC_UART0->RBR;                     /* lee el dato recibido y lo almacena */
            if (*ptr_rx++ ==13)                    // Caracter return --> Cadena completa
                        {
                        *ptr_rx=0;/* Añadimos el caracter null para tratar los datos recibidos como una cadena*/
                        rx_completa = 1;/* rx completa */
                        ptr_rx=buffer; /* puntero al inicio del buffer para nueva recepción */
                        }
                break;


        case 0x02:                                  /* THRE, Transmit Holding Register empty */
        if (*ptr_tx!=0) LPC_UART0->THR=*ptr_tx++;   /* carga un nuevo dato para ser transmitido */
        else tx_completa=1;
        break;

    }
}

// Función para enviar una cadena de texto
// El argumento de entrada es la dirección de la cadena, o
// directamente la cadena de texto entre comillas

void tx_cadena_UART0(char *cadena)
{
ptr_tx=cadena;
tx_completa=0;
LPC_UART0->THR=*ptr_tx++;      // IMPORTANTE: Introducir un carácter al comienzo para iniciar TX o
}                              // activar flag interrupción por registro transmisor vacio
```