UNER Facultad de Ingeniería

Trabajo Práctico VI

Fundamentos de Programación de Sistemas Embebidos
Especialización en Sistemas Embebidos
Cohorte 2018

Objetivos

- Analizar la aplicación de arreglos, estructuras y uniones en la resolución de problemáticas comunes de sistemas embebidos.
- Adquirir destreza en el uso e implementación de estructuras de datos simples.

Herramientas requeridas

Para el desarrollo del presente trabajo práctico deberá contar con:

- Equipo PC utilizando Linux como sistema operativo.
- Paquete de compilación de GNU GCC para ARM incluyendo binutils-arm-none-eabi, gcc-arm-none-eabi y make.
- Open OCD
- Eclipse IDE for GNU ARM (versión recomendada Oxygen 3a)
- Placa de desarrollo EDU-CIAA NXP.

1. Simón dice...

El juego *Simón dice* o *sigan al lider* consiste en que uno de los jugadores realiza un movimiento y los demás jugadores deben copiarlo. Si alguno de los jugadores no lo imita de forma adecuada o demora su respuesta, pierde.

La versión electrónica de este juego consiste en un dispositivo con cuatro luces de colores (rojo, amarillo, azul y verde) que se encienden utilizando una secuencia aleatoria. Al comenzar el juego se enciende una luz y el jugador debe presionar en el pulsador asociado al mismo. Si presiona en el pulsador correcto pasa a la siguiente etapa, donde se encienden dos luces y el jugador debe pulsar en el orden correcto los pulsadores asociados. Según contesta correctamente, se agrega una luz a la secuencia y se disminuye el tiempo de encendido de cada luz. Si el jugador presiona un pulsador que no se corresponde con la secuencia indicada, pierde el juego.

Actividades:

Implemente una versión electrónica del juego *Simón dice* utilizando los pulsadores y los leds incorporados en la EDU-CIAA NXP. Documente la aplicación utilizando `doxygen`.

Para esto considere:

- El juego se realizará considerando un total de 10 niveles, por lo que la secuencia final se corresponderá con el encendido de 10 leds en un orden particular al que deberá responderse con la misma secuencia de pulsación en los actuadores.
- Si el jugador gana deberá parpadear el led verde y si pierde el led rojo, 5 veces a intervalos de 250 ms.
- Previo al inicio del juego se deberá representar una secuencia lumínica en los leds de la placa. Para esto deberá utilizar las interrupciones del SysTick. Al presionar un pulsador y luego de una pausa de 500 ms, deberá comenzar el juego encendiendo el primer led y esperando se presione el pulsador asociado.
- En cada pulsación se deberá encender el led asociado al pulsador.
- El control de los pulsadores deberá realizarse utilizando interrupciones.

2. Pila

Una de las estructuras de datos clásicas es la conocida como pila o `stack`. Esta estructura de datos se caracteriza por tener un comportamiento denominado LIFO (*Last Input - First Output*), esto se corresponde con la idea de 'apilar' valores uno encima del otro, de modo tal que al quitar un valor se tomará el último agregado.

Este comportamiento logra que al cargar un conjunto de valores y luego extraerlos se obtenga la versión reflejada del conjunto.

Actividades:

1. Realice una implementación de una estructura de datos de tipo pila. Es de interés que sea posible trabajar con diferentes 'pilas' en la misma aplicación y que cada uno pueda manejar diferente cantidad de elementos y elementos de diferente tamaño. Considere las siguientes definiciones como base de la implementación:

```
/*
  Estructura definida para representar cada pila particular. Nótese que el
  vector de almacenamiento se establece a través de la función pilaInit
  y debe ser definido, de forma externa a la estructura y de tipo congruente
  con los valores a alojar en la pila.
*/
typedef struct
{
  uint8_t pos;       /* Posición donde cargar el proximo valor */
  uint8_t dataSize;  /* Tamaño de cada dato en el arreglo  buf */
  uint8_t dataCount; /* Tamaño del buffer                      */
  uint8_t *buf;      /* Buffer de almacenamiento               */
} pilaData;

void    pilaInit(pilaData *p,     /* Representación de una pila       */
                 void *buffer,    /* Vector de almacenamiento         */
                 uint8_t bufLen,  /* Tamaño del arreglo               */
                 uint8_t dataSize /* Tamaño de cada dato en el arreglo*/
                 );
void    pilaPush (pilaData *p, void *data);
void    pilaPop  (pilaData *p, void *data);
uint8_t pilaEmpty(pilaData *p);
uint8_t pilaFull (pilaData *p);
```

   Implemente y documente las funciones anteriores utilizando `doxygen`.

2. En el ejercicio 2 de la guía de trabajos prácticos 1 se pedía implementar una función que, utilizando `putchar`, mostrara la representación de un valor entero. Uno de los problemas de esta implementación reside en que si se toma consecutivamente el resto de la división por 10 para identificar cada dígito, se tiene el orden inverso de los valores.

   Utilizando las funciones desarrollas para gestionar una pila, realice una nueva implementación de una función que obtenga la representación de un número de 32 bits sin signo y la almacene en una cadena de caracteres.

   Utilice la implementación anterior para enviar un valor entero a través de la USART2 y de este modo verificar el correcto funcionamiento del algoritmo.

3. ¿Cuál es el tamaño óptimo para la definición del arreglo que contendrá los dígitos en la implementación de la función del punto anterior?

4. Modifique el programa desarrollado en el ejercicio 2 de la guía de trabajos práctico 1 de modo que la salida estándar se de a través de la USART2. Analice la complejidad y el tamaño del binario logrado en ambos casos.

3. Comunicación basada en paquetes

Una forma muy común de implementar comunicaciones entre un sistema embebido y un equipo PC es utilizando una estructura de paquetes. En estos casos se envía una sucesión de valores y, si estos cumplen con cierta estructura, producen una acción en el sistema embebido y/o una respuesta.

Una forma simple de implementar estar forma de comunicación consiste en construir paquetes a partir de un encabezado y finalización constante, la información objeto de la comunicación y una suma de comprobación.

Considerando un paquete de 5 bytes, podría estructurar se como:

b0	b1	b2	b3	b4
0x0C	cmd	target	suma	0xC0

Donde:

- `b0`: constituye el encabezado constante igual al valor $0C_{16}$
- `b1`: representa la operación a realizar. Valores de interés son $0F_{16}$ para encender un led y $F0_{16}$ para apagarlo.
- `b2`: número de led objetivo, siendo válidos identificadores de 1 a 6 (leds 1 a 3 y los tres colores del led RGB).
- `b3`: suma de comprobación, la suma de todos los valores (a excepción de este mismo campo) como un valor de 8 bits.
- `b4`: constituye la marca de fin de los datos, es un valor constante igual a $C0_{16}$

Actividades:

- Implemente una aplicación que permita recibir a través de la `USART2` un flujo de información como el indicado y obre en consecuencia encendiendo o apagando un led particular. Documento el desarrollo utilizando `doxygen`.

  Para esta implementación utilice las estructuras de datos siguientes:

```
#define dataLen 5

typedef struct
{
  uint8_t start;
  uint8_t cmd;
  uint8_t target;
  uint8_t crc;
  uint8_t end;
} dataPckt __attribute__ ((packed));

typedef uint8_t dataBytes[dataLen];

typedef union
{
  dataPckt st;
  dataBytes db;

}packet;
```

  La implementación deberá recibir paquetes con el formato indicado y, si el mismo resulta inválido, no realizar ninguna operación. Considere ocasionalmente puede enviarse un valor

incorrecto, pero a continuación de este puede seguir una secuencia correspondiente a un paquete válido y el sistema deberá entenderlo de forma correcta.

Compruebe el correcto funcionamiento de la aplicación utilizando `gtkTerm`, para simplificar las pruebas active el envío de valores hexadecimales desde `View/Send hexadecimal data`.

4. Código Morse

El *Código Morse* es una forma de codificación que permite el envío de información utilizando sonidos o luces y un alfabeto constituido por puntos o rayas. Ambos se representan por la duración en la que está encendida una fuente de luz o que dura un sonido determinado. Asumiendo que la duración de un 'punto' es tomada como referencia, la duración de una 'raya' es tres veces la duración de un punto.

Cada letra y los dígitos se representan utilizando una secuencia de rayas y/o puntos (Cuadros 1 y 2). Entre cada uno de ellos, debe existir una pausa equivalente a la duración de un punto. El espacio entre dos letras consecutivas debe ser igual a tres puntos. Finalmente, el espacio entre palabras debe ser igual a la duración de 5 puntos.

Letra	Representación	Letra	Representación
A	`P R`	N	`R P`
B	`R P P P`	O	`R R R`
C	`R P R P`	P	`P R R P`
D	`R P P`	Q	`R R P R`
E	`P`	R	`P R P`
F	`P P R P`	S	`P P P`
G	`R R P`	T	`R`
H	`P P P P`	U	`P P R`
I	`P P`	V	`P P P R`
J	`P R R R`	W	`P R R`
K	`R P R`	X	`R P P R`
L	`P R P P`	Y	`R P R R`
M	`R R`	Z	`R R P P`

Cuadro 1: Código Morse de las letras del alfabeto. Se utiliza `P` para representar un 'punto' y `R` para representar una 'raya'

Digito	Representación
0	R R R R R
1	P R R R R
2	P P R R R
3	P P P R R
4	P P P P R
5	P P P P P
6	R P P P P
7	R R P P P
8	R R R P P
9	R R R R P

Cuadro 2: Código Morse de los dígitos del 0 al 9.

Actividad:

Desarrolle una aplicación que, según se envíen oraciones, logre la representación de la misma utilizando código Morse y los leds presentes en la EDU-CIAA NXP. Documente la aplicación utilizando `doxygen`.

Considere para la implementación:

- Se utilizará la USART2 para recibir la información y ninguna frase contendrá más de 250 caracteres.
- La duración del punto, base de la codificación, puede tomar diferentes valores. Utilice como valor de base una duración de 300 ms. Defina una constante simbólica para este valor y para representar la duración de la raya, espera entre símbolos, espera entre letras y espera entre palabras.
- Realice la traducción de texto ASCII a código Morse utilizando una tabla de consulta alojada en memoria FLASH. Para esto considere definir arreglos por cada símbolo marcando el final de los mismos con el valor cero. Luego defina un vector que en cada posición contenga los vectores definidos anteriormente. Realice estas definiciones tanto para los símbolos alfabéticos como para los numéricos.
- Una vez desarrollada la aplicación, verifique a través de la dirección de los arreglos, que se hayan alojado en memoria flash.