Práctica 3: Juego Mastermind con consola serie

3.1. Objetivo de la práctica

En esta práctica está dividida en dos partes. En la primera parte afianzaremos los conocimientos adquiridos en la práctica 2 sobre el funcionamiento del teclado matricial, los temporizadores y el display 8-segmentos. En la segunda parte añadiremos un dispositivo nuevo, la UART, con el que añadiremos funcionalidad al proyecto desarrollado en la primera parte.

Los principales objetivos de la práctica son:

- Comprender la problemática del diseño de sistemas más complejos con dispositivos manejados por interrupciones.
- Conocer los conceptos básicos de la comunicación serie asíncrona.
- Familiarizarse con la unidad UART del S3C44BOX.

3.2. Primera Parte: Implementación del juego Mastermind

En esta primera parte de la práctica vamos a implementar una versión simplificada del juego *Mastermind*. En el juego *Mastermind* un jugador pone una clave (en nuestro caso de 4 dígitos), que el adversario deberá adivinar (en nuestro caso, no pondremos límite de intentos). En el juego original, tras cada intento se recibe *feedback* por parte del usuario que puso la clave indicando cómo se acercaba nuestro intento a la clave real. En nuestra práctica sólo obtendremos dos posibles respuestas: clave correcta o incorrecta.

Para su desarrollo usaremos el teclado matricial tanto para introducir la clave como para los sucesivos intentos. También usaremos el display 8-segmentos y el timer 0. Para ello tendremos que usar todos los ficheros utilizados en la Práctica 2, excepto el main.c, que será el fichero que se tendrá que implementar.

3.2.1. Descripción del funcionamiento del sistema

La figura 3.1 muestra el flujo de ejecución que debe tener la aplicación. Detallemos paso a paso:

Al comenzar la ejecución, el display 8-segmentos mostrará una letra `C` indicando que está esperando la introducción de la clave.

El usuario comenzará a introducir la clave en el teclado matricial. Podrá hacer tantas pulsaciones como desee, pero sólo se tendrán en cuenta las 4 anteriores a pulsar la tecla `F`. Los dígitos introducidos deben almacenarse en un *buffer* denominado `passwd`. Este *buffer* será un *array* que se gestionará de forma que siempre almacene las últimas cuatro pulsaciones. La clave consta de 4 teclas, por lo que será necesario haber pulsado al menos 4 teclas antes de pulsar la `F`. Si el número de teclas pulsadas antes de pulsar la `F` fuese menor que 4 se producirá un error y en el display se mostrará una `E`. El sistema sigue esperando a que se introduzcan 4 teclas de la clave.

Cuando el usuario pulse la tecla `F` se mostrará la clave introducida por el display 8-segmentos (hasta este momento, el display únicamente mostraba `C`). Deberá mostrar un **dígito**

por segundo y para ello **debe usarse el timer 0** (no debe hacerse con espera activa, por ejemplo utilizando la función `Delay()`).

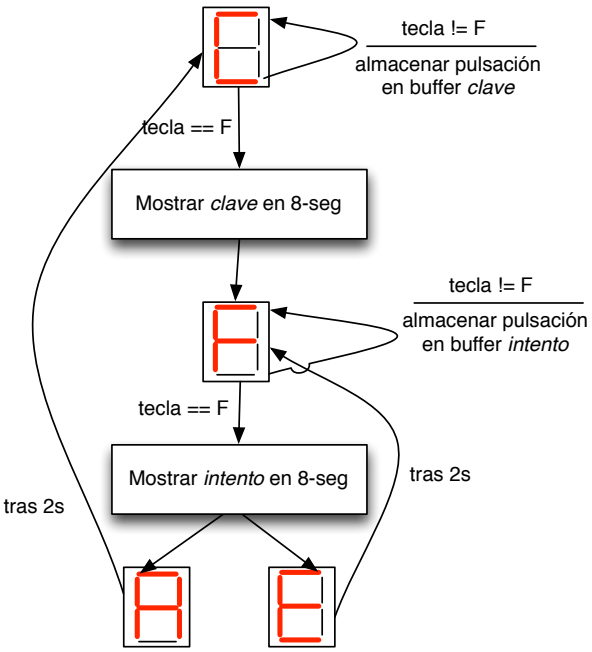

Figura 3.1: Diagrama de flujo de la aplicación

Tras el paso anterior, el display 8-segmentos mostrará una `F` para indicar que se puede proceder a tratar de adivinar la clave.

Nuevamente, el usuario realizará pulsaciones (mínimo 4) y acabará pulsando la tecla `F`. Si el número de teclas pulsadas antes de pulsar la `F` fuese menor que 4 se producirá un error y en el display se mostrará una `E`. El sistema sigue esperando a que se introduzcan 4 teclas de la clave. Los dígitos introducidos deben almacenarse en un *buffer* denominado `guess` de modo que, al finalizar este paso, el intento de 4 dígitos esté almacenado en dicho buffer.

Cuando el usuario pulse la tecla `F` se mostrará el intento introducido por el display 8-segmentos (hasta este momento, el display mostraba `F`). Deberá mostrar un **dígito por segundo** y para ello **debe usarse el timer 0** (de nuevo no debe hacerse mediante espera activa con la función `Delay()`).

Tras el paso anterior, el sistema mostrará durante dos segundos el resultado del juego por el display de 8 segmentos. Si se acertó con la clave se mostrará una `A` y si no, se mostrará una `E`. La espera de dos segundos se deberá hacer también mediante interrupciones del timer 0. Finalmente, si se acertó la clave el juego volverá a empezar. Si no, se volverá a mostrar una `F` en espera de nuevos intentos.

3.2.2. Implementación del sistema

Es habitual de modelar este tipo de sistemas como una máquina de estados. El estado de la máquina se representa como una variable global. El programa principal espera siempre a que se produzca un *evento*. Cuando se produce el evento decide lo que hay que hacer en función del valor de la variable de estado y vuelve a bloquearse a la espera de que se produzca otro evento. Esto se repite indefinidamente.

En esta práctica utilizaremos la máquina de estados mostrada en la figura 3.2 para modelar el sistema descrito en la sección **??**. Dicha máquina tiene 5 estados posibles:

- **INIT**. Estado inicial, desde el que comienza el juego. En este estado la máquina muestra el dígito 12 (C) por el display de 8 segmentos y el usuario puede introducir la clave a través del teclado matricial, que será almacenada en el *buffer* **passwd**. En este estado se espera al evento *clave introducida*, que se produce cuando el usuario pulsa la tecla 15 (F). Cuando se produce este evento la máquina comprueba si la clave introducida tiene al menos 4 digitos. En caso afirmativo pasará al estado `SPWD`, en caso negativo

se quedará en el mismo estado, mostrará el dígito `E` en el display de 8 segmentos y se quedará de nuevo esperando a que el usuario introduzca otra vez una clave.

- **SPWD**. Estado mostrar clave (*show password*). La máquina configura el timer 0 para mostrar la clave almacenada en el display de 8 segmentos dígito a dígito, cambiando de dígito en intervalos de 1s. La máquina se quedará entonces esperando el evento *fin de la visualización* disparado por la rutina de tratamiento de interrupción del timer. Al recibir el evento la máquina pasará al estado `DOGUESS`.

- **DOGUESS**. Estado de intento de adivinar contraseña. Este estado es similar a `INIT`, la máquina muestra 15 (F) por el display 8 segmentos y el usuario puede introducir un intento de adivinar la clave por el teclado matricial. Al pulsar la tecla 15 (F) se generará el evento *clave introducida* que hará que la máquina compruebe si es válida (4 dígitos o más). Si lo es, la máquina pasará al estado `SGUESS`, y si no, mostrará por el display de 8 segmentos el dígito E y volverá a quedarse esperando a que el usuario introduzca un nuevo intento.

- **SGUESS**. Estado mostrar intento (*show guess*). Este estado es similar al estado `SPWD`, la máquina prepara el timer para la visualización del intento almacenado y esperará a que se genere el evento *fin de la visualización*. Al recibir el evento pasará al estado `GOVER`

- **GOVER**. Estado final, en el que se mostrará el. La máquina comparará las dos claves almacenadas, si coinciden mostrará en el display de 8 segmentos el dígito 10 (A) durante 2 segundos, y si no coincide mostrará el dígito 14 (E). El sistema esperará al evento *fin de la visualización* y pasará al estado `INIT` si fue un acierto y al estado `DOGUESS` si fue un error.

Para facilitar la implementación de esta máquina usaremos las funciones:

- `void read_kbd(char *buffer)`: esta función prepara la ISR de gestión del teclado para que almacene las teclas pulsadas en `buffer`. Para ello se utilizan dos variables globales:

  - `char *keyBuffer`, y
  - `int keyCount`.

  La primera será inicializada con el valor `buffer` y la segunda será puesta a 0. Después `read_kbd` habilitará las interrupciones por `EINT1` y se quedará esperando hasta que la ISR del teclado ponga a `NULL` la variable `keyBuffer` (lo que representa la llegada del evento *clave introducida*). Es decir, al invocar la función `read_kbd` el programa se quedará esperando hasta que se termine de introducir una clave. Al volver, la función devolverá `keyCount`, que tendrá la cuenta de los dígitos introducidos.

- `void printD8Led(char *buffer, int size)`: esta función prapara la ISR del timer 0 para que muestre por el display de 8 segmentos los `size` dígitos almacenados en `buffer`. Para ello utilizará dos variables globales:

  - `char *tmrBuffer`
  - `int tmrBuffSize`

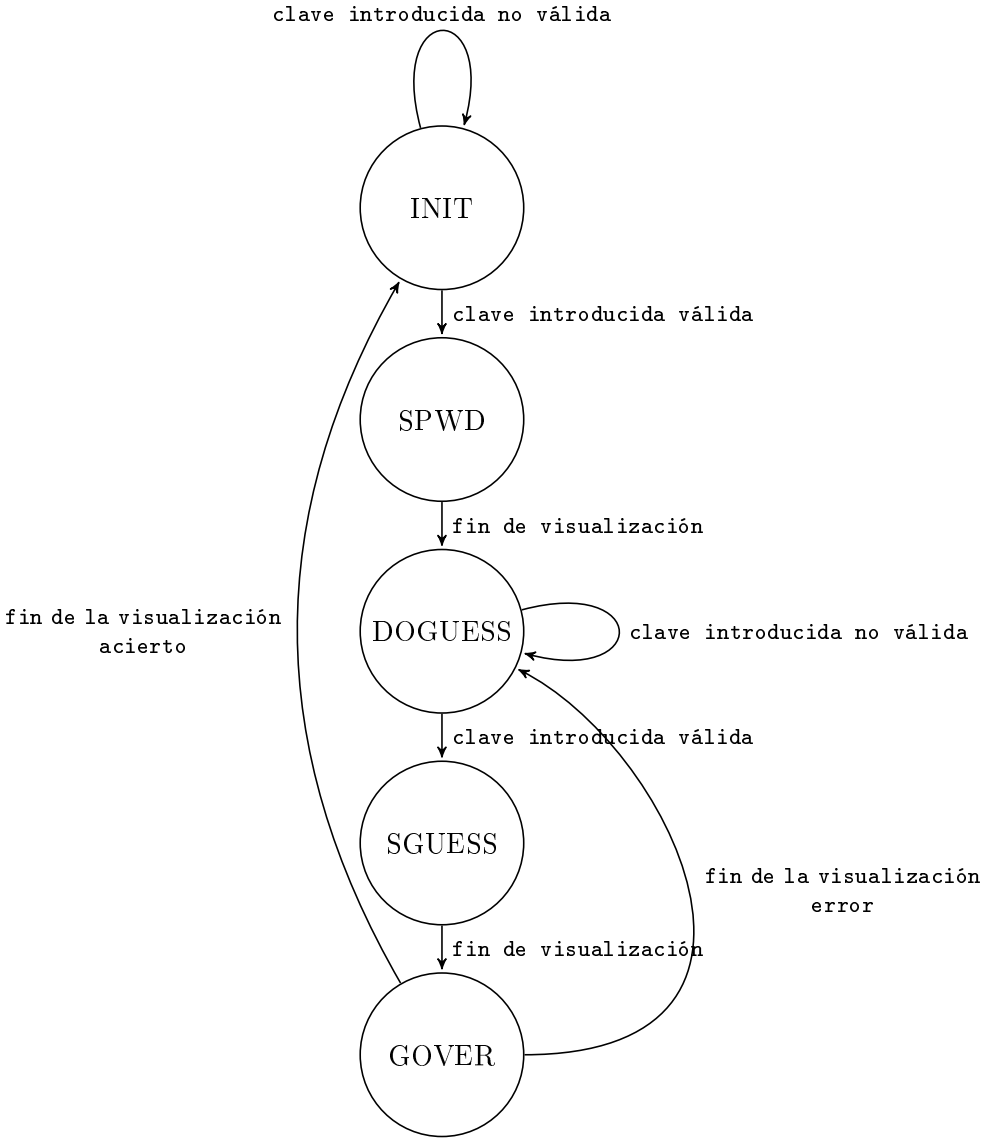

Figura 3.2: Diagrama de estados de la aplicación

La primera será inicializada con el valor `buffer` y la segunda será inicializada con el valor `size`. Después iniciará el timer y se quedará esperando a que la ISR del timer ponga `tmrBuffer` a `NULL`, indicando con ello el *fin de la visualización*.

- `void show_result()`: esta función prepara la ISR del timer 0 para que muestre por el display de 8 segmentos una cadena con dos dígitos (AA en caso de acierto y EE en caso de error) y se quedará esperando a que termine la visualización. Esto lo hace como `printD8Led` utilizando el puntero `tmrBuffer` y la variable `tmrBufSize`. El efecto neto que se consigue con esto es que se muestre A ó E durante 2 segundos. Como en el caso de `printD8Led` la función quedará esperando hasta que la ISR del timer ponga a `NULL` el puntero `tmrBuffer`.

Las correspondientes rutinas de tratamiento de interrupción (ISRs), desactivarán la generación de interrupciones correspondientes, enmascarando la línea `EINT1` en el caso del teclado y parando el timer en el caso del timer, además de poner los punteros a `NULL` .

Con la ayuda de estas funciones la máquina de estados se correspondería con el pseudocódigo del cuadro 1.

3.2.3. Desarrollo de la primera parte de la práctica

Para completar la práctica debemos incluir los ficheros de los módulos desarrollados en la práctica 2 (utils, button, D8Led, intcontroller, gpio, leds, keyboard y timer) además de los ficheros init.asm y 44b.h. Añadiremos también el fichero main.c suministrado por el profesor. Este fichero está incompleto. El alumno debe completar el código de main siguiendo las explicaciones de este guión y las indicaciones de los comentarios del propio código.

3.3. Segunda Parte: añadiendo una consola serie

En esta segunda parte de la práctica vamos a dotar al juego de una consola serie, de forma que en la fase de juego (estado `DOGUESS`) se solicite al jugador la contraseña por un terminal serie, conectado al puerto serie de la placa. Para ello podemos utilizar cualquier programa de terminal serie de nuestro sistema (Hyperterminal, Termite, Minicom, etc.).

En el equipo de laboratorio tenemos instalado `Termite`, para utilizarlo tendríamos que abrirlo y seleccionar el puerto serie virtual ofrecido por el driver de ftdi (que maneja el chip ftdi del *dongle* Olimex conectado a la placa). Además deberemos configurar el puerto con los mismos parámetros que configuremos la uart del s3c44b0x, que son:

- 115200 baudios
- 8 bits de datos
- Sin paridad
- Un bit de parada
- Sin control de flujo

Cuadro 1 Pseudocódigo de la máquina de estados

```
int loop (void)
{
    int cont;

    switch (gstate){
        case INIT:
            do {
                Visualizar C en el display;
                count = read_kdb(passwd);
                if count<4
                    visualizar una E;
            } while (count < 4);
            sig. estado;
            break;

        case SPWD:
            printD8Led(passwd,N);
            sig estado;
            break;

        case DOGUESS:
            do {
                Visualizar F en el display;
                count = read_kdb(guess);
                if (count < 4)
                    visualizar una E;
            } while (count < 4);
            sig. estado;
            break;

        case SGUESS:
            printD8Led(guess,N);
            sig estado;
            break;

        case GOVER:
            error = show_result();
            if (error)
                sig estado = DGUESS;
            else
                sig estado = INIT;
            break;
    }
}
```

Una vez realizada la conexión del terminal, se recomienda al alumno resetear la placa pulsando el botón de reset. Si la configuración del terminal serie se ha realizado correctamente debe aparecer en el terminal un menú similar al que se muestra en el display LCD de la placa, con el que podemos interaccionar con el programa de test residente en ROM.

Para dar soporte a la consola serie debemos hacer dos cosas:

1. Modificar el programa principal para que en el estado `DOGUESS` lea del puerto serie 0 (UART0) en lugar de leer del teclado matricial.
2. Implementar el módulo uart, del que el profesor suministrará un esquema incompleto que el alumno deberá completar tras leer las indicaciones del guión y siguiendo los comentarios del código.

En las siguientes secciones se describen en detalle estas modificaciones.

3.3.1. Modificaciones en el programa principal

En el estado `DOGUESS` la lectura del puerto serie se hará mediante la función:

```
readline(char *buffer, int size)
```

que reemplazará a la función `read_kbd` que se utilizó en la primera parte, y se utilizará el buffer `readlineBuf` como argumento. El alumno debe completar la implementación de esta función, cuya misión es leer una línea del terminal serie copiándola en el `buffer` pasado como argumento. Una línea no es más que una cadena de caracteres terminada en `'\r'`. La función `readline` por tanto leerá caracteres hasta encontrar `'\r'` o hasta llenar el `buffer` que se le pasa (tamaño indicado por `size`), lo que antes suceda. La función debería devolver una cadena de caracteres C bien formada por lo que en realidad debe reservarse una posición en el `buffer` para escribir el final de cadena, marcado con el carácter `nul` (`'\0'`, un byte a 0). Para leer cada carácter se debe utilizar la función:

```
int uart_getch(enum UART port, char *c);
```

implementada en el módulo uart (fichero `uart.c`). Consultar el fichero `uart.h` para ver los valores posibles del parámetro `port`.

Una línea será una contraseña válida si su tamaño es de 4 o más caracteres. Una vez tengamos una línea válida, copiaremos los últimos 4 dígitos en el buffer `guess`, transformándolos previamente de ASCII a decimal. Recordemos que del puerto serie leemos los códigos ASCII de las teclas pulsadas en el terminal, es decir, si pulsamos la tecla 0 se enviará el valor 0x30 (48) y no el 0x0 (0), que es lo que hemos utilizado para representar la tecla 0 en el caso del teclado matricial. Para hacer esta transformación se proporciona la función `int ascii2digit(char c)`. Se aconseja examinar el código de esta función.

3.3.2. Implementación del módulo uart

Para gestionar los puertos serie, el módulo uart debe asociar cierta información de estado a cada puerto. Esta información se almacena en el array de dos elementos `uport`, siendo cada elemento del array una estructura `port_stat` declarada como sigue:

```
struct port_stat {
```

```
        enum URxTxMode rxmode;      //Modo de recepción (DIS, POLL, INT, DMA)
        enum URxTxMode txmode;      //Modo de envío     (DIS, POLL, INT, DMA)
        unsigned char ibuf[BUFLEN]; //Buffer de recepción (usado en modo INT)
        int rP;                     //Puntero de lectura en ibuf (modo INT)
        int wP;                     //Puntero de escritura en ibuf (modo INT)
        char *sendP;                //Puntero a la cadena de envío (modo INT)
        enum ONOFF echo;            //Flag para echo de caracteres recibidos
};
```

El interfaz del módulo uart se compone de las siguientes funciones:

- `void uart_init(void)`, función de inicialización de módulo, debe ser invocada antes de utilizar el módulo. Principalmente inicializa los campos de las estructuras `port_stat` de cada puerto.

- `int uart_lconf(enum UART port, struct ulconf *lconf)`, función de configuración del modo de línea de la uart. Permite seleccionar la velocidad del puerto (baudios), el número de bits de datos, el número de bits de parada, la paridad y si se habilita o no el modo infrarrojos. Recibe la configuración deseada como parámetro (estructura `ulconf`) y la función escribe los valores adecuados en los registros `ULCONn` y `UBRDIVn`. Además la función configura adecuadamente los pines de los puertos E y C que sacan las señales `rx` y `tx` de los dos puertos fuera del chip a los conectores DB9 de la placa.

- `int uart_conf_txmode(enum UART port, enum URxTxMode mode)`, función que permite seleccionar el modo en el que actuará la uart en transmisión: polling, interrupciones o dma.

- `int uart_conf_rxmode(enum UART port, enum URxTxMode mode)`, función que permite seleccionar el modo en el que actuará la uart en recepción: polling, interrupciones o dma.

- `int uart_getch(enum UART port, char *c)`, función bloqueante (síncrona) que lee un byte del puerto serie, y lo escribe en la dirección que se pasa como argumento (`c`).

- `int uart_sendch(enum UART port, char c)`, función bloqueante (síncrona) que envía el carácter que se pasa como argumento (`c`) por el puerto serie seleccionado.

- `int uart_send_str(enum UART port, char *str)`, función bloqueante (síncrona) que envía la cadena de caracteres apuntada por `str` por el puerto serie seleccionado.

- `void uart_printf(enum UART port, char *fmt, ...)`, función que envía por el puerto serie una cadena de caracteres con formato `fmt`, al estilo de printf (emplea `uart_send_str`). Esta función se da completamente implementada.

Las funciones de recepción y envío, se comportan de forma diferente en función del modo en que se haya configurado el puerto. Esta información está almacenada en la estructura `port_stat` correspondiente al puerto.

En la recepción por *polling* la función `uart_getch` debe quedarse consultando el registro de estado del puerto serie hasta que se active el flag de recepción. Entonces leerá del registro buffer de recepción el carácter recibido y lo devolverá.

En la recepción por interrupciones, ilustrada en la **??**, la función `uart_getch` se quedará esperando hasta que haya algún carácter en el buffer de recepción asociado al puerto (campo

`ibuf` de la estructura `port_stat`). El buffer de recepción se gestiona de forma circular, utilizando un puntero de lectura (campo `rP`) y un puntero de escritura (campo `wP`). La función `uart_getch` leerá del buffer mediante la función `uart_readfrombuf` en la posición indicada por el puntero de lectura, y hará avanzar dicho puntero. Los caracteres son insertados en el buffer de recepción por la rutina de tratamiento de interrupción (RTI) de recepción. Cada vez que se recibe un carácter la RTI almacena el carácter leído en el buffer en la posición indicada por el puntero de escritura. Cuando el puerto se configura para funcionar en recepción por interrupciones la línea de interrupción correspondiente debe quedar activa.

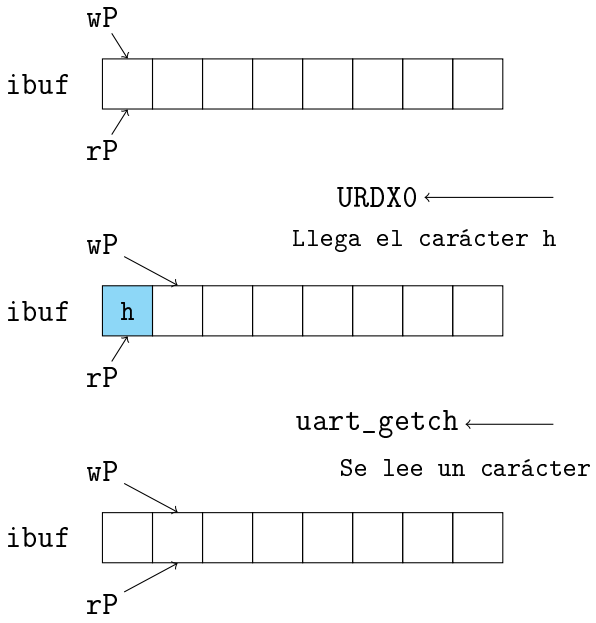

Figura 3.3: Recepción uart por interrupciones

En el envío por *polling* la función `uart_sendch` debe esperar leyendo el registro de estado del puerto serie hasta que se marque el flag que indica que el registro de buffer de envío está vacío. En ese momento puede escribir el carácter a enviar en el registro y retornar.

En el envío por interrupciones, la función `uart_send_str` debe inicializar el campo `sendP` de la estructura `port_stat` del puerto con la dirección de comienzo de la cadena de caracteres que hay que enviar. Luego habilitará las interrupciones por envío y se quedará esperando a que la RTI de envío ponga `sendP` a `NULL`, indicando que el envío se ha completado. Por su parte la RTI se ejecutará cada vez que se vacíe el registro buffer de envío, y copiará entonces en dicho registro el carácter apuntado por `sendP`, incrementando después el puntero. Cuando el puntero queda apuntando al final de cadena (carácter `nul`), la propia RTI deshabilita la línea de interrupción por envío de la uart y pone el puntero `sendP` a `NULL` para indicar el final del envío.

Y finalmente, en el envío por interrupciones la función `uart_sendch` utilizará `uart_send_str` para enviar una cadena de un sólo carácter. Mientras que en el envío por *polling* la función `uart_send_str` utilizará `uart_sendch` para enviar uno a uno los caracteres de la cadena que hay que enviar.

3.3.3. Desarrollo de la segunda parte de la práctica

Para completar esta parte de la práctica, el profesor suministrará a los alumnos 6 ficheros: `util.c`, `util.h`, `uart.c`, `uart.h`, `44b.h` y `main.c`, que deberán añadirse a los ficheros de los módulos desarrollados en la práctica anterior para completar el proyecto. Sólo dos de los nuevos ficheros han de ser completados por los alumnos, `uart.c` y `main.c`, siguiendo las indicaciones dadas en este guión y los comentarios insertados en los propios ficheros.