TRABAJO MICROCONTROLADOR

MSP430G22553

Parking Assistant

SERGIO LEÓN DONCEL

ÁLVARO GARCÍA LORA

SISTEMAS ELECTRÓNICOS

3º GIERM

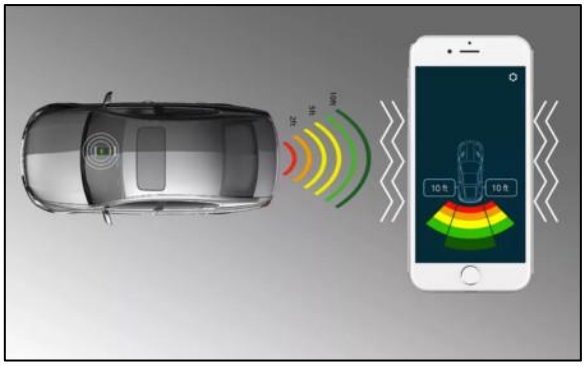
Índice

Introducción. Presentación del proyecto

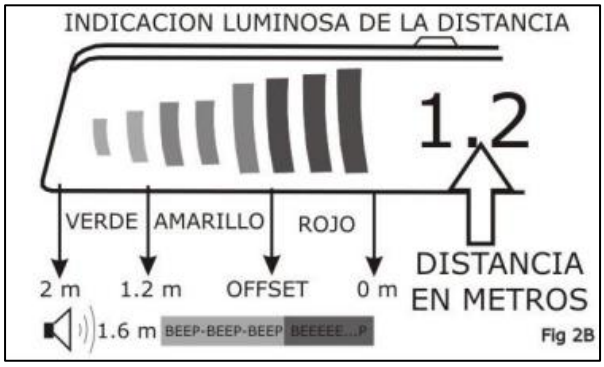
Descripción de idea:

La maniobra de aparcamiento es una de las más complejas, y provoca dificultades para muchos conductores, a pesar de ser una de las más cotidianas e importantes. Esto se debe a la poca visibilidad del habitáculo del vehículo y la escasa información de la que disponemos sobre nuestro alrededor para decidir como actuar. Sin embargo, si disponemos de los sensores adecuados, una pantalla con los datos necesarios, y una señal auditiva que nos alerte, sería mucho más sencillo aparcar sin cometer errores.

A través del microcontrolador MSP430 gestionamos una interfaz visual con una pantalla LCD de resolución 128x128, la toma de mediciones con 2 sensores HC-SR04 (delantero y trasero), un indicador auditivo con buzzer, y el control por distancia del sistema a través de smartphone con conexión Bluetooh Classic mediante el módulo de transmisión y recepción bluetooth HC-06.



Esquema de funcionamiento del sistema



Propuesta de interfaz por pantalla

De esta forma, tendremos en todo momento información de la distancia entre el vehículo y el obstáculo, y estaremos alerta para no colisionar ya que dispondremos de señal auditiva intermitente que varía según la proximidad.

Para hacer su uso más universal se utiliza un modo de configuración por el que el usuario puede navegar y retocar los ajustes. En el mismo, se puede elegir entre el idioma español e inglés. Además, es posible también elegir el sistema de unidades usado para expresar el valor de la distancia: centímetros o pulgadas.

También se desea que la interfaz sea amigable con el usuario, de tal manera que se puede elegir entre una oscura o clara según el gusto del consumidor / luminosidad que necesite en cada momento.

Para ser lo más versátil posile y de nuevo pensando en la experiencia de usuario, hay ocasiones donde los pitidos pueden ser más una distracción que una ayuda. Por tanto, se podrá elegir si activar o desactivar el sonido, ya que de todas maneras se dispondrá de indicadores visuales en forma de barra que muestra cómo de cerca nos encontramos del obstáculo.

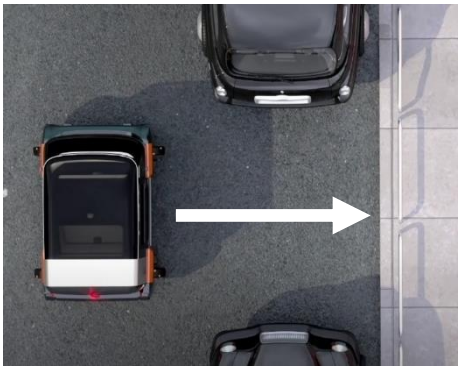
Se ha tenido también en cuenta la seguridad al volante. No se puede permitir que mientras estamos conduciendo haya distracciones, por tanto, sólo se permitirá acceder a la configuración si el coche ha sido previamente parado.

Por último, sería muy tedioso tener que volver a configurar nuestros gustos cada vez que apaguemos y encendamos el coche. Es por ello que se ha decidido guardar en el segmento D de la flash los últimos valores de configuración asignados. Así, esta se mantendrá como estaba a pesar de que se deje de alimentar al microcontrolador.

Extensiones a otras aplicaciones:

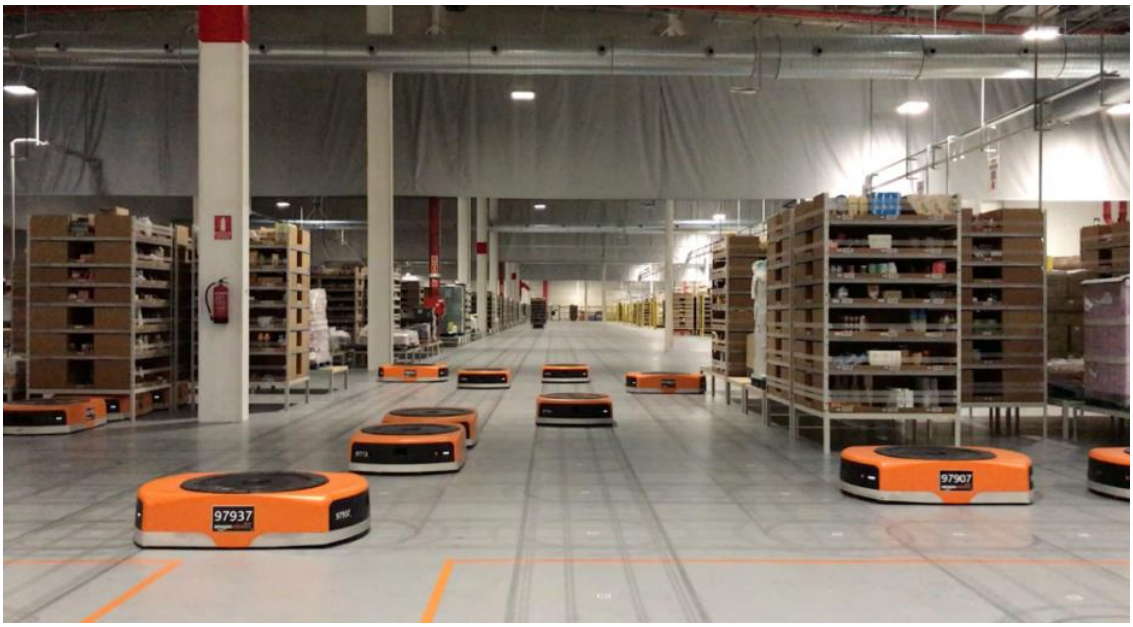
Poder aparcar el coche con nuestro sistema ya dejará de ser un problema. Sin embargo, no es su única utilidad. Existe un novedoso sistema de dirección que permite girar las 4 ruedas simultáneamente, denominado por los fabricantes "Crab Walk" o "Modo Cangrejo", en el cuál el coche se desplazaría lateralmente. En ese caso, podría usarse este sistema para controlar la distancia lateral y así facilitar este tipo de aparcameiento.

Sistema dirección "Crab Walk"



Ventaja del aparcamiento lateral

El sistema es muy versátil, se podría utilizar en múltiples aplicaciones. Por ejemplo, en los almacénes logísticos de grandes instalaciones, sería un bloque fundamental de los AMR (robots móviles autónomos) para la medida de la distancia.



AMRs de Amazon en uno de sus centros logísticos

Otro uso más sencillo sería el de su implementación en drones o dispositivos RC (radiocontrol), donde si se requisiera de precisión, no valdría tan solo con tener una cámara que nos muestre el entorno, sino que se requeriría de información de distancias para posteriormente decidir como controlarlo remotamente gracias a la comunicación Bluetooth incorporada.

Además, dada la limitada capacidad del MSP430, se ha decidido implementar tan solo en 2 idiomas y con sólo 2 modos de interfaz. No obstante, con un microcontrolador de la misma familia que dispusiera de más memoria seríamos capaces de tener en cuenta muchísimos más idiomas para permitir un uso más global, permitir al usuario elegir el color principal en pantalla, tipología de fuente, etc. Al disponer de más pines utilizar más sensores para asegurar una misma medida recibida y evitar problemas como el reflejo con el suelo.

Incluso se podría extender al nivel de desarrollo de los REX (Robots Explorer), dado que para su movimiento autónomo se necesita conocer en todo momento la distancia de este con los objetos de su alrededor.

Fases y coordinación de tareas

El proyecto puede ser divido en distintas fases de desarrollo, las cuales sirven de guión para lograr el funcionamiento general deseado. En cada una de ellas se afronta un objetivo específico, que junto a las demás completará todo el sistema.

- Medición de distancia

  En esta primera fase nos informamos acerca del funcionamiento del sensor ultrasónico HC-SR04 con lectura del datasheet. Planteamos como abordar la generación y recepción de las señales por los pines que cuenta el sensor para conseguir obtener una primera medida de distancia. Una vez desarrollado el código se llevó a cabo un periodo de depuración para detectar errores y corregirlos. Una vez se consigue el funcionamiento se llevan acabo numerosas pruebas para comprobar la correcta medida de la distancia usando para ello diferentes objetos a modo de obstaculos y comparando la medida con una regla. Una vez superada con éxito esta fase se incluyó otro sensor ultrasónico y se procedió a modificar el código para conseguir una alternancia entre ambos sensores, de tal forma que fueramos capaces de elegir entre uno u otro con vistas futuras a usarlos como delantero y trasero del vehículo.

- Alarma sonora

  De forma independiente a la medición de medidas previamente lograda, se realizó un código en el cual controlabamos el buzzer simulando una medida tomada. Previamente se pensó en como hacer sonar la alarma que avisa al conductor, defiendo una lógica en cuanto a la relación entre distancia, intermitencia y volumen se refiere. Hicimos numerosas pruebas ajustando los valores de DC(duty cycle) del PWm desarrollado para este propósito y conseguir el sonido buscado, haciendolo más intuitivo, menos molesto y adaptado al conductor en definitiva. Una vez logrado esto, acoplamos la alarma a la primera fase de medición de la distancia, consiguiendo así que en función de la distancia medida por los sensores, el buzzer sonara de una u otra forma.

- Conexión bluetooth

  Para abordar esta fase, nos informamos sobre el protocolo de comunicación bluetooth y el módulo HC-06. Primeramente aprendimos a configurar el dispositivo mediante comandos AT, para de esta forma asegurarnos de que tuviera una velocidad de transmisión de datos de 9600 baudios. Una vez añadido fisicamente al circuito eléctrico y tras haber realizado el conexionado necesario fuimos dandole las primeras órdenes de control del sistema para implementar los modos de movimiento del vehículo y así dar paso a un sensor ultrasónico u otro. Para ello usamos la app de 'Bluetooth Electronics' en la cual fuimos montando el cuadro de mandos a lo largo del proyecto, para ir satisfaciendo las necesidades que nos iban surgiendo.

- Procesado de información y representación en pantalla principal
  En esta fase adaptamos la información de medida proporcionada por los sensores de ultrasonidos a una pantalla principal, en la cuál se muestra diferentes formas y mensajes en función de la situación en la que se encuentre el vehículo.
- Configuración y ajustes
  Esta parte consistió en la realización del menú de configuración y su control mediante el cuadro de mandos de la aplicación del móvil. Pensamos en varias opciones interesantes que podrían dotar al sistema de una mayor versatilidad y adaptación personalizada.
- Orden y pruebas de funcionamiento
  Por último, una vez abarcados todos los objetivos planteados en un primer momento, procedimos a unir cada una de las partes todas bajo el mismo código, de nuevo con fase de depuración hasta encontrar los errores, subsanarlos y conseguir el funcionamiento completo deseado.
- Desarrollo de orden y estructuración de código
  Tras haber obtenido el funcionamiento global del sistema, dedicamos una fase de estructurado de código en base a funciones con un orden de jerarquía, de forma que el código fuera mucho más claro y legible.
- Fase de optimización
  Aunque ya se hubiera tenido en cuenta en fases previas el consumo de energía y la optimización de uso de memoria flash y ram en el programa, se decidió dedicar una jornada de mejora y búsqueda de la eficiencia para el sistema.

En lineas generales, se ha trabajado de forma conjunta para todas las fases. Primero cada uno hacía un planteamiento a papel de qué se deseaba obtener y luego se esbozaba una primera idea de cómo se debía resolver. Tras ello, desarrollabamos cada uno nuestro planteamiento, comparabamos y decidíamos cúal era mejor para implementarlo en el programa. La fase de programación, depuración y optimización se ha llevado en líneas generales en sesiones conjuntas, de forma que en todo momento mientras alguien escribía código, la otra persona se encargara de ofrecerle mejoras, corregirle fallos o buscar otra manera de abordarlo si no se conseguía avanzar por un camino.

Solución adoptada

Uso de periféricos

Los periféricos internos usados han sido los siguientes:

- Pines:
  En los siguientes apartados se detallan los pines usados, configuración y conexión con el hardware implementado.

- Timer:
  Hemos usado los dos temporizadores del Timer A.

  TIMER0A: Configurado a 1 MHz, en modo up/down y con módulo máximo (65536), conseguimos un periodo por encima de 60 ms (65,5 ms aproximados). Este periodo es necesario puesto que el timer se usa para generar una señal PWM de las características apropiadas para el correcto funcionamiento de los sensores de ultrasonido (usamos comparador 1 para dar un duty cycle muy bajo como indica el fabricante del sensor de ultrasonido). La pin P1.6 el cual está conectado a los trigger de los sensores HCSR-04.

  TIMER1A: Configurado a 1 MHz, en modo continuo, por tanto el módulo es el máximo del contador (65536). Se consigue el mismo periodo que con el TIMER0A (65,5 ms aproximados).
  Su uso tiene tres objetivos distintos:
  - Conseguir cazar los flancos de subida y bajada en los pines P2.1 y P2.2 asociados a los echo de los sensores ultrasonidos. Para ello usamos el modo Capture/Compare con el comparador 1 activando ambos flancos en la cofiguración. De esta forma obtenemos el tiempo existente entre un flanco de subida y un flanco de bajada en los echo, lo cual es necesario para poder saber el valor de la distancia, como se detalla en el apartado de implementación de hardware. En función del movimiento del vehículo seleccionamos o bien el echo del HC-SR04 delantero o trasero mediante el cambio del selector de entrada del modo Capture/Compare.
  - Generar una señal PWM junto con el comparador 2 para el uso del buzzer. En función de la distancia al obstáculo cambia el duty cycle para conseguir un efecto de control de volumen en el buzzer.
  - Medir tiempo para conseguir una intermitencia de on/off en el buzzer, de tal forma que a menor distancia mayor frecuencia en la intermitencia exista, indicando así el peligro al usuario.

- PWM:
  Resumiendo, las señales PWM usadas:
  - TIMER0A + comparador 1 → Señal para trigger
  - TIMER1A + comparador 2 → Señal para buzzer
- Interrupciones:
  Contamos con 3 rutinas de interrupción:
  - Interrupción del TIMER1A + comparador 1 : Nos posibilita el cálculo de la distancia. La interrupción salta cuando se da un flanco en el echo del sensor de ultrasonido que se encuentre activo.
  - Interrupción del TIMER1A + comparador 0: Es usada para contar tiempos con aplicación en la intermitencia del buzzer. La interrupción salta cada perido del temporizador (65,5 ms aprox).
  - Interrupción de USCI-A (modo UART): Sirve para recibir los comandos enviados vía bluetooth. La interrución salta cada vez que se manda algún comando.

- USCI-A (UART):
  Hacemos uso de la USCI-A en modo UART para establecer una comunicación serie entre el módulo bluetooth y el micro. Ajustamos para ello los registros UCA0BR0 y UCA0BR1 para conseguir los 9600 baudios de transmisión de datos (módulo bluetooth previamente configurado a la misma velocidad).

- Flash:
  Utilizamos un segmento de memoria flash para almacenar la configuración que se ha dado en el menú de ajustes al sistema por parte del usuario (volumen, idioma, unidad de medida y color de interfaz)

- LPM:
  Usamos el modo de bajo consumo principalmente para 2 objetivos. El primero de ellos es la eficiencia. Se ha deseado tomar medidas a una frecuencia de 15 Hz (cada 65ms), por lo que el resto del tiempo no hay nada que ejecutar ni actualizar en pantalla, por tanto, para ahorrar energía se manda a dormir el micro. Además, en ciertas partes del código convenía realizar esperas hasta que ocurriera determinado evento (interrupción), por lo que se optaba por esta solución.

Pines utilizados

- P1.1 (Rx)
  - Conectado al Tx del módulo bluetooth HC-06.
  - Configurado como como entrada y para su uso con la USCI-A en modo UART como receptor de datos.
- P1.2 (Tx)
  - Conectado al Rx del módulo bluetooth HC-06.
  - Configurado como salida y para su uso con la USCI-A en modo UART como transmisor de datos.

- P1.6 (Trigger)
  - Conectado a los *trigger* de los sensores HCSR-04.
  - Configurado como salida y en modo PWM asociado con el timer 0 comparador 1.

- P2.1 (Echo delantero)
  - Conectado al *echo* del sensor HCSR-04 usado como medidor de distancia delantero (activo cuando el vehículo está en marcha).
  - Configurado como entrada. Alterna su modo entre I/O y PWM asociado con el timer 1 comparador 1.
- P2.2 (Echo trasero)
  - Conectado al *echo* del sensor HCSR-04 usado como medidor de distancia trasero (activo cuando el vehículo está en retroceso).
  - Configurado como entrada. Alterna su modo entre I/O y PWM asociado con el timer 1 comparador 1.
- P2.4 (Buzzer)
  - Conectado con el buzzer.
  - Configurado como salida. Cambia su modo de funcionamiento entre I/O (siendo salida a off) y PWM asociada con el timer 1 comparador 2, dependiendo en cada caso si el buzzer debe sonar o no.

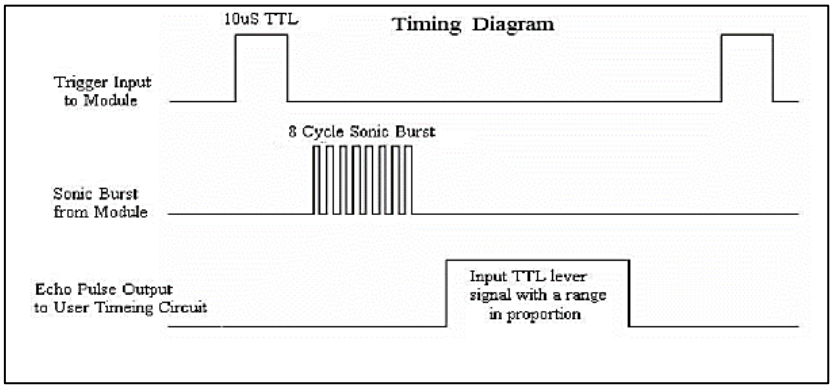
- P1.5 , P1.7, P2.0, P2.3 y P2.7 (Pantalla Boosterpack)
  - Pines asociados a la pantalla del boosterpack.

Hardware adicional implementado

- Sensores de ultrasonido HC-SR04:

Conseguimos hallar las distancias en el aparcamiento del vehículo gracias al principio de funcionamiento del HC-SR04, quien mediante la emisión y recepción cíclica de un impulso acústico de alta frecuencia. El impulso se propaga a la velocidad del sonido por el aire y al encontrar un obtáculo, revota, siendo reflejado y vuelviendo como eco al sensor ultrasónico. La distancia hacia al obstáculo podemos calcularla basandonos en el tiempo que transcurre entre la emisión de la señal (por el pin de trigger) y la recepción de la señal (por el pin de echo).

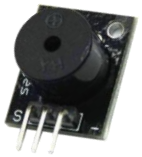
A continuación se muestra un diagrama de tiempos de operación del sesor, sacado del datasheet proporcionado por el fabricante.



Como se ha descrito en el apartado de pines, usamos el pin P1.6 para generar una señal PWM que esté a valor alto durante 10 us y a valor bajo el resto del periodo (conectada al trigger de los sensores). El fabricante nos recomienda que dicho periodo sea superior a 60 ms, de ahí nuestra elección a la hora de hacer la configuración de los timers. Los pines P2.1 y P2.2, como ya se ha expuesto se usan para los pines echo de los sensores. El intervalo de tiempo transcurrido en us entre flanco de subida y bajada es el usado para el cálculo de la distancia mediante las siguientes relaciones (obtenidas teniendo en cuenta la velocidad de propagación del sonido):

Distancia (cm) = intervalo (us)/58
Distancia (plg) = intervalo (us)/148

- Buzzer:



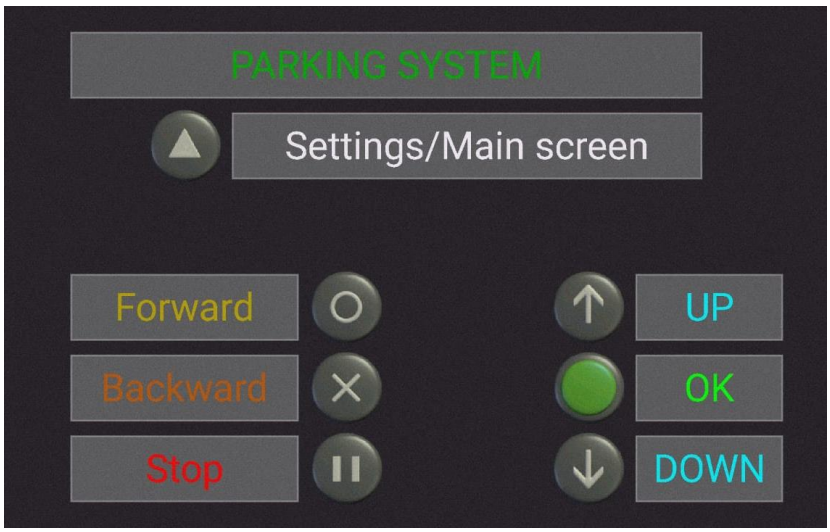
Cumple con el fin de avisar al conductor de la proximidad de otro vehículo de forma sonora. El pin positivo o de señal del buzzer está conectado al pin P2.4 quien funciona como PWM cuando se desea que el actuador emita sonido o salida digital baja cuando se requiera que esté silenciado (bien por la alternancia que se da en el pitido jugando con ella en función de la distancia o bien porque el usuario cambie a modo silencio el sistema mediante el menu de configuración). La reducción o incremento del volumen en cada caso se consigue con la modificación del duty cycle.

La lógica que se ha seguido es la siguiente: a mayor distancia, menor frecuencia de intermitencia y menor volumen. A menor distancia, por tanto, mayor frecuencia de intermitencia y mayor volumen avisando del peligro inminente al conductor. Si la distancia es lo sufientemente grande no se emite ningún sonido, lo cual es sinónimo de seguridad.

- Módulo Bluetooth HC-06:

Para comunicarnos con el microcontrolador y enviarle ordenes para controlar el sistema hacemos uso del módulo bluetooth HC-06. Mediante la app 'Bluetooth Electronics' disponible para Android, hemos diseñado un cuadro de mandos para enviar comandos al módulo, el cuál transmite dicha información al micro por la UART. Hemos configurado para ello las velocidades de ambos a 9600 baudios.
Los pines Tx y Rx del módulo HC-06 se conectan a los pines P1.1 (Rx) y P1.2 (Tx) del launchpad.

A continuación se muestra el cuadro de mandos, desarrollado en la app mencionda anteriormente, al cual haremos referencia en la explicación del menú de configuración que se trata en el siguiente punto.



- Pantalla del Boosterpack:
  La información que genera el sistema de medidas se muestra al usuario por la pantalla del boosterpack, quien mediante un menu de configuración es capaz de adecuar los ajustes a sus necesidades personales. A continuación se detallarán las distintas opciones con las que cuenta la pantalla:

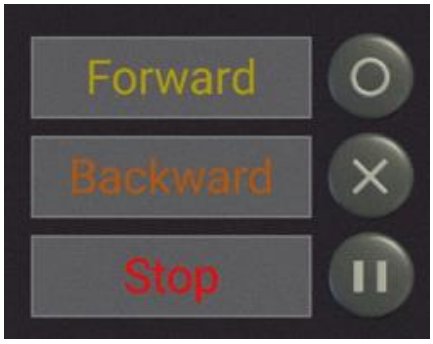
  - *Pantalla espera:*
    El sistema se mantiene en espera hasta que se realiza con éxito la conexión bluetooth.

- *Pantalla principal*:
  Aquí es donde se muestra al usuario la información relativa a la distancia estructurada en distintas partes:

  - *Barras indicadoras de proximidad:*
    A medida que nos vamos acercando se van activando cada vez más barras de mayor tamaño y color más calido, indicando una mayor cercanía al obstáculo. En el caso en que la distancia sea consideramente lejana, no existe ninguna barra. El otro caso límite sería el peligro inminente representado por las 6 barras activadas.

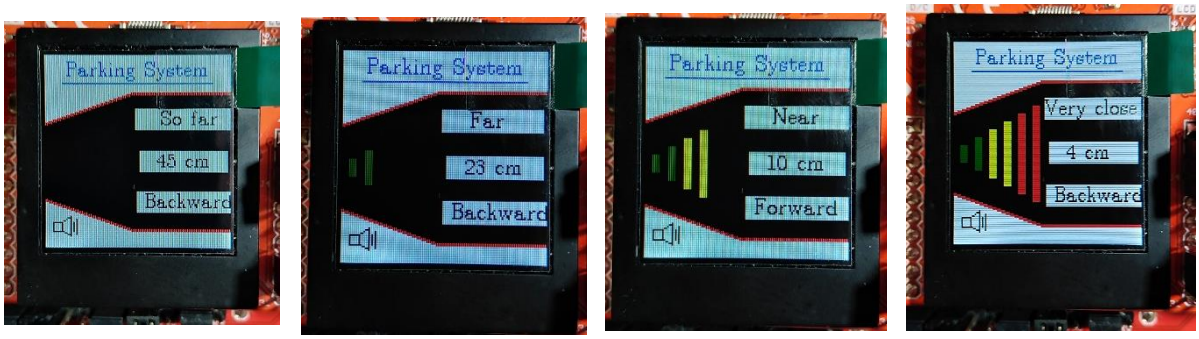
  - *Valor de la medida:*
    Valor de la distancia tomada por el sensor delantero o trasero según el conductor este avanzando o retrocediendo.

  - *Indicador del movimiento del vehículo:*
    Un recuadro debajo del valor de la medida indica en cada instante si el vehículo se encuentra parado, en marcha (avanzando) o en retroceso. Simulamos el movimiento del coche con el cuadro de mandos:

  - *Mensaje de aviso:*
    Un recuadro encima del valor de la medida nos avisa con diversos mensajes que se corresponden con la cercanía o lejanía al obstáculo.

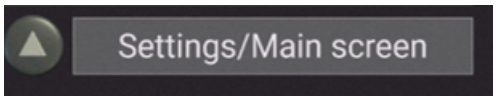
  - *Icono de sonido:*
    Un simbolo en la esquina inferior izquierda muestra si el sonido está activado (y por tanto el buzzer también) o si de lo contrario está muteado.



- *Pantalla de configuración* :

  (a diferencia de los tres botones de marcha, parada y retroceso que son para simulación, los que vienen a continuación estarían disponibles para el conductor)

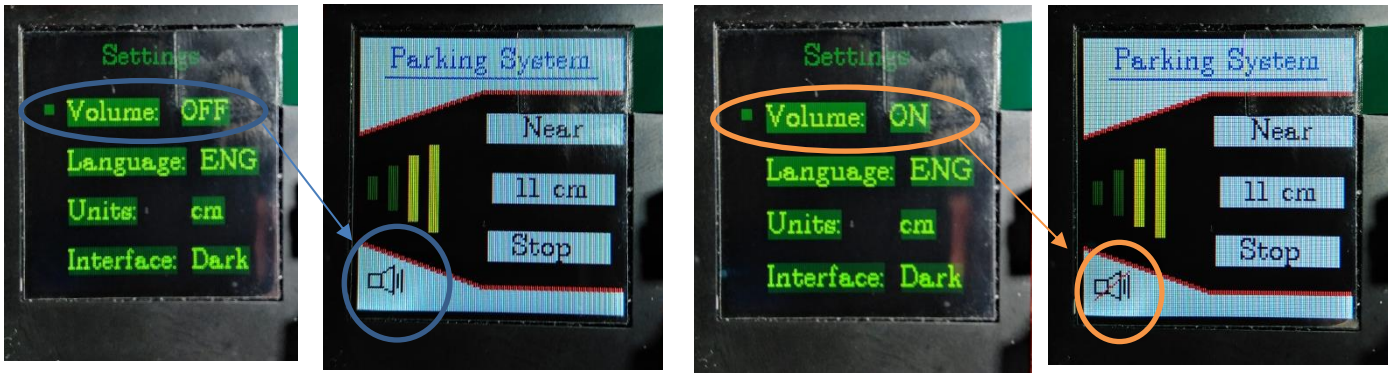
  Para acceder a la pantalla de configuración, por razones de seguridad al volante, el coche debe estar parado. Si es así, el usuario entraria en este menú de ajustes mediante la pulsación del siguiente botón en el cuadro de mandos:



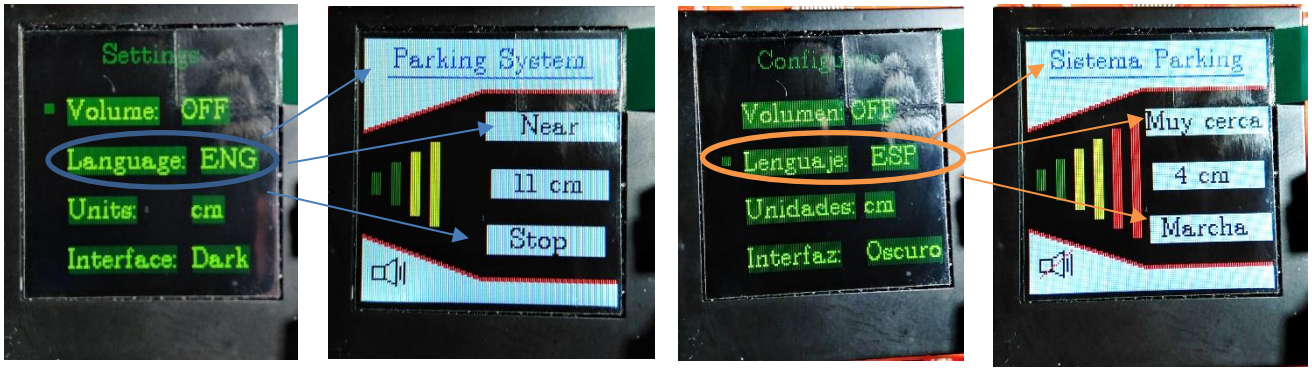
  Para salir de nuevo a la pantalla principal se usa el mismo botón (mismo comando bluetooth asociado).

  Para moverse por las distintas opciones de la configuración usamos las flechas de up/down del cuadro de mandos. Para cambiar algún ajuste basta con presionar el botón de OK.

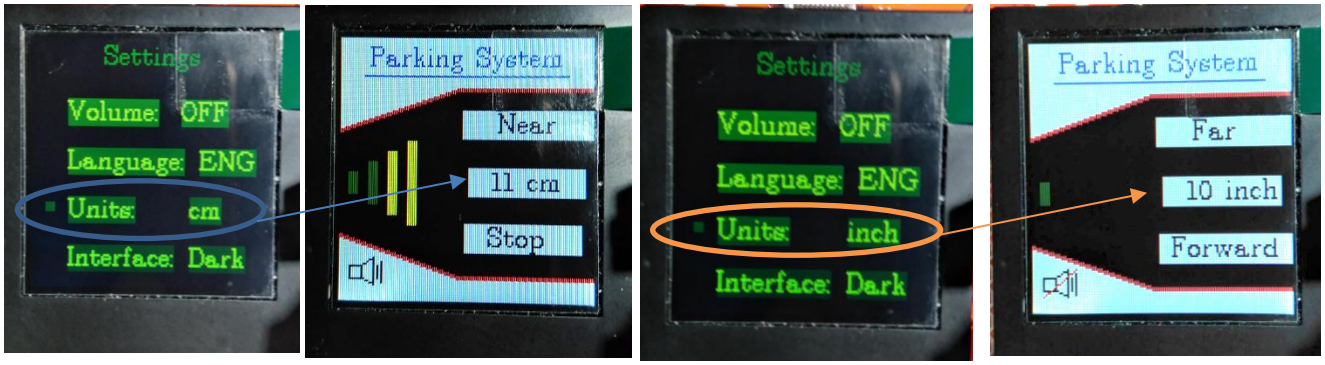
- *Volumen: ON / OFF*



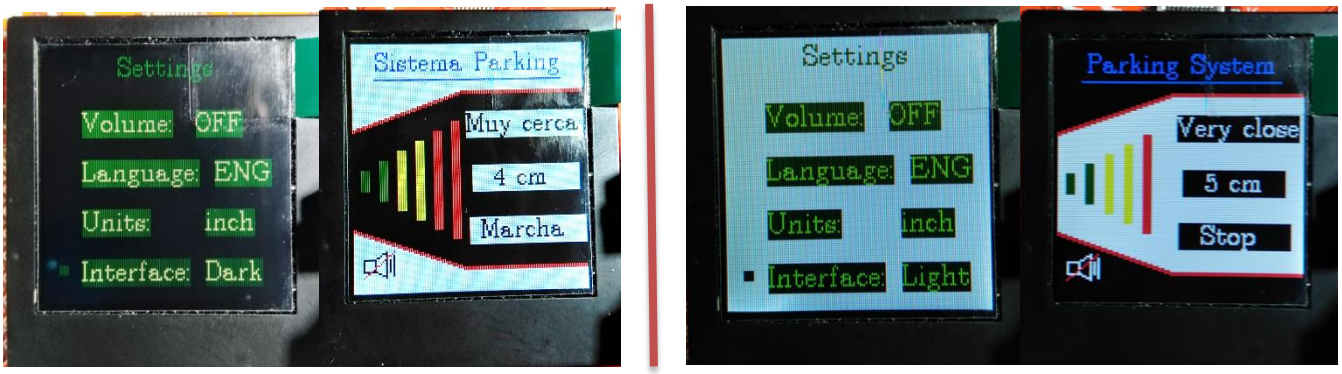
- *Idioma* : Inglés / Español



- *Sistema de unidades* : Centímetros / Pulgadas

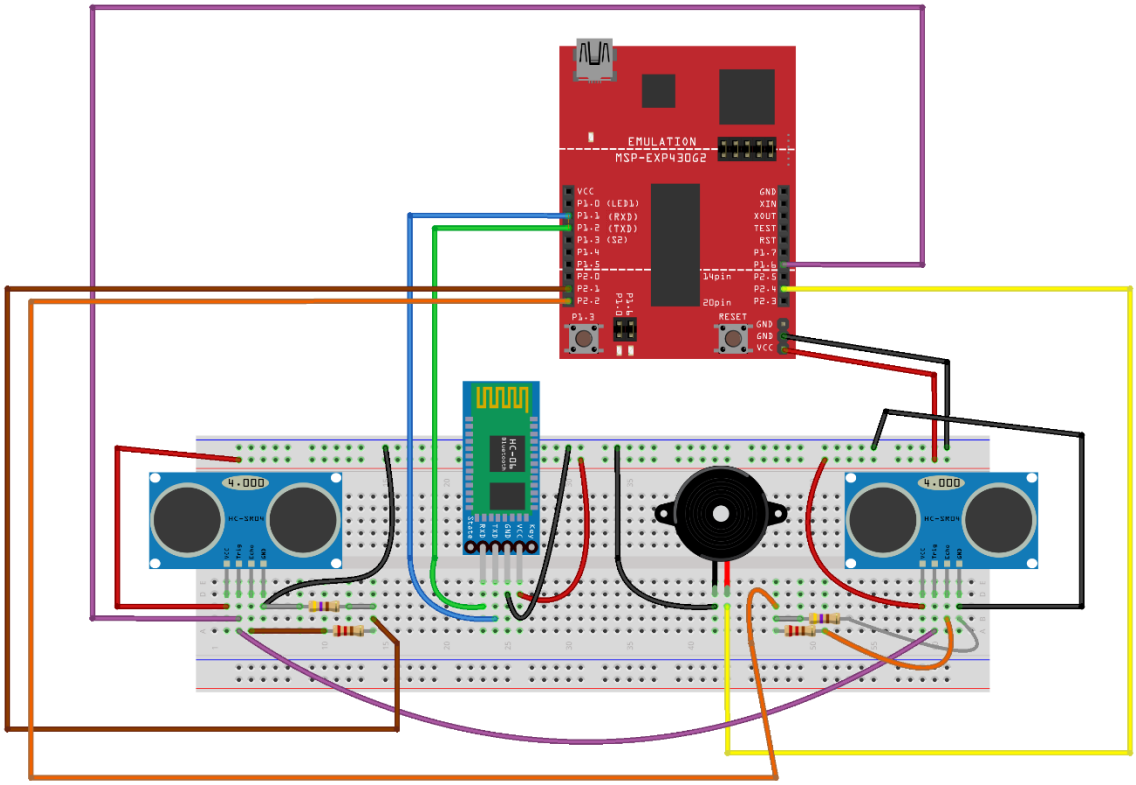


- *Color Interfaz* : Modo oscuro / Modo claro



Esquemas eléctricos de montaje

Conexionado de pines del launchpad con los distintos componentes usados:



Explicación de código

El programa se encuentra organizado de forma que tiene una estructura principal (función main), a través de la cuál se controla qué realiza el micro en función de las llamadas a otras funciones secundarias. Por tanto, comenzaremos explicando el main:

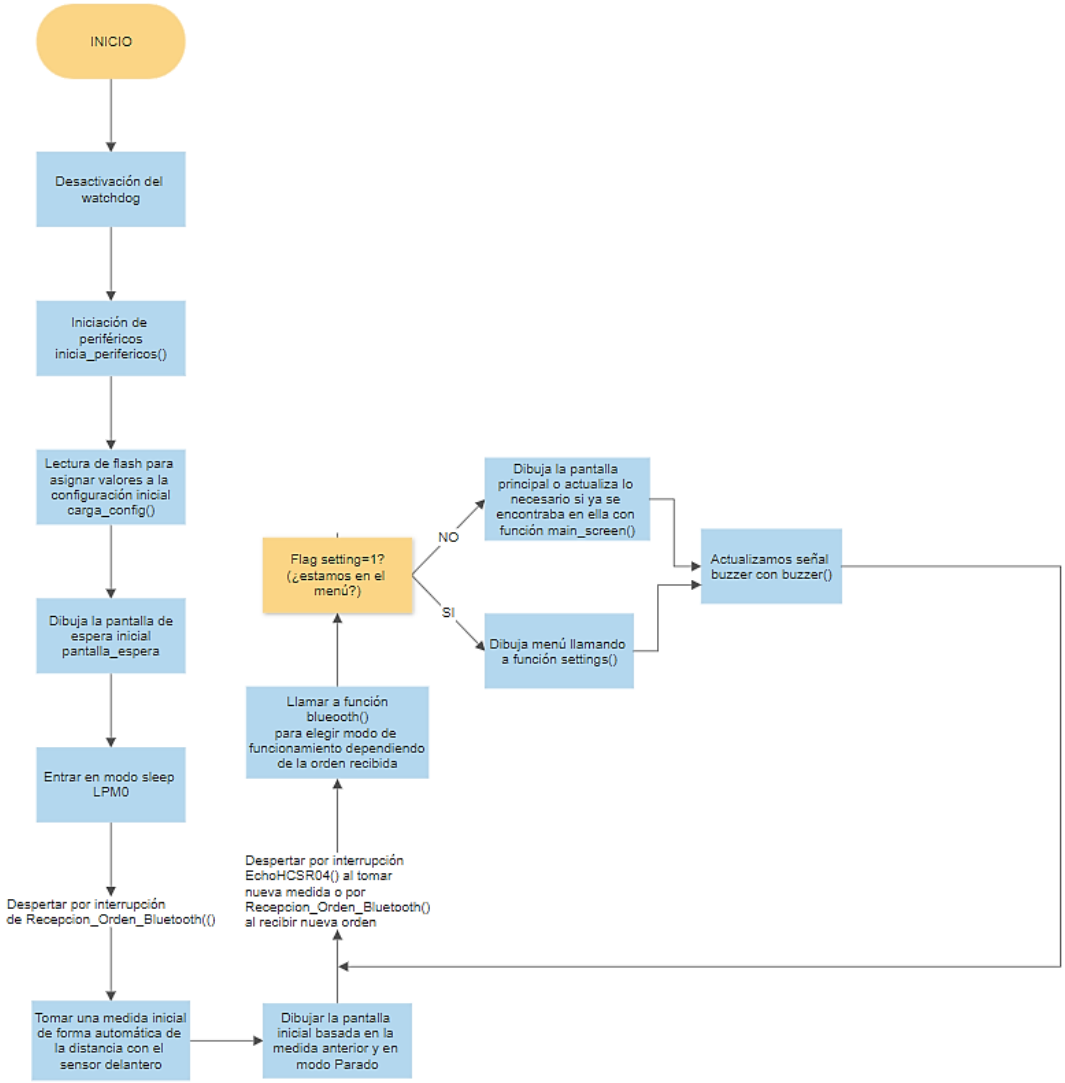


Diagrama de flujo del main

Lo primero de todo será describir las interrupciones que se pueden producir:

-EchoHCS04(): Encargada de tomar mediciones cada 65 ms (frecuencia de 15Hz) captando el tiempo donde se produce flanco de subida y de bajada de la señal echo como se explicaba en apartados anteriores, y despertar al micro. Guarda en el vector tempo los 2 valores de tiempo donde se producen los flancos, calcula la diferencia en diff y guarda la distancia en distancia_cm y distancia_plg utilizando la fórmula obtenida del datasheet.

-Recepcion_Orden_Bluetooth(): Simplemente guarda el valor del registro UCA0RXBUF (que contiene la orden recibida por bluetooth, es decir, un carácter) en la variable comando_bluetooth, pone a 1 la bandera FinRx, que indica que se ha recibido una orden nueva, y despierta al micro.

-Contador_Buzzer(): Su única función será la de incrementar la variable t cada 65 ms de forma que podamos temporizar cuánto tiempo lleva activado o desactivado el PWM del buzzer.

Para conseguir ejecutar el algoritmo previamente mostrado en el diagrama, necesitaremos definir cada una de las funciones anteriormente nombradas:

-inicia_perifericos(): Utiliza la función Set_Clk() para configurar la frecuencia del reloj SMCLK. Se configuran los pines 1.1(RX) y 1.2(TX) como entrada y salida respectivamente para la USCI-A, el pin 1.6 como trigger de ambos sensores como PWM con el timer 0 y comparador 1, el pin 2.1 como echo del delantero y 2.2 del trasero (con uso del timer 1 comparador 1), pin 2.4 como salida PWM asociado al timer 1 comparador 2 (inicialmente lo dejamos apagado) para el manejo del buzzer.

Además se utilizan los timers con módulo de 65ms y velocidad de 1MHz (para cumplir con la restricción del fabricante de dejar espacio de 60 ms entre medida y medida) y se configura la pantalla.

Seguidamente, se ajusta la comunicación serie por UART a una velocidad de 9600 baudios (para hacerla coincidir con la del móulo bluetooth HC-06) gracias a los registros UCA0BR0 y UCA0BR1, ya que concadenando uno con otro se elige el divisor de frecuencia a utilizar (en nuestro caso 833, para que al dividir 8M entre dicho número se obtenga aproximadamente esos 9600 baudios buscados).

Por último, se divide entre 24 la frecuencia para que la velocidad de lectura de la flash sea la correcta y no se tengan problemas con ella, y se han activado las interrupciones globales.

-primera_medida: Llamando a la función control_timers() se selecciona el sensor delantero para tomar una medida de distancia, esperamos 3 periódos de medida para asegurar que la medida sea la correcta, y apagamos nuevamente los timers y el sensor utilizados.

La función control_timers() se utiliza para elegir en función del parámetro sel y encendido cuál de ellos se desea encender o si se desea apagar los sensores.

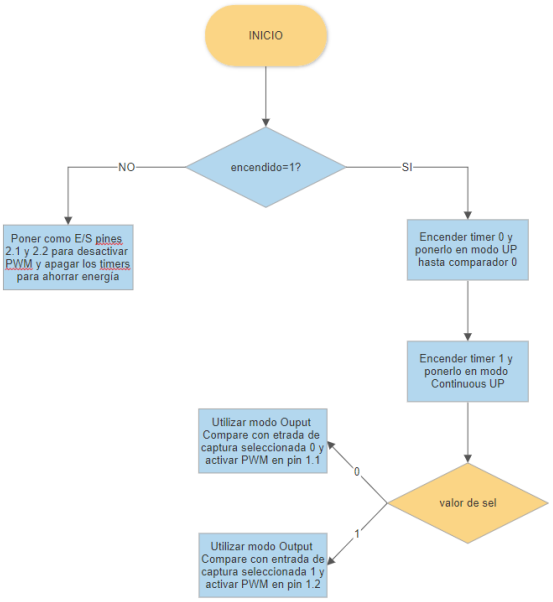


Diagrama de flujo de Control_timers()

- Funciones uart_putc(), uart_puts() y uart_putfr() que se utilizan para mostrar frases por terminal.

-pantalla_espera(), pinta_fondo(), pinta_titulo(), dibuja_altavoz(), borra_barras() y dibuja_barras() son funciones principalmente gráficas, donde se utliza las funciones gráficas de la librería grlib.h para mostrar en pantalla lo deseado.

-bluetooth() se encarga de comprobar el valor de la bandera FinRx para ver si se ha mandado una nueva orden. En caso afirmativo, actua en función del valor de la orden recibida, bien activando el sensor correspondiente, apagándolo, o alternando el valor de setting para cambiar entre pantalla principal y de configuración o al revés. Si se trata de una orden de control de mandos activa la bandera correspondiente (okey, up, down).

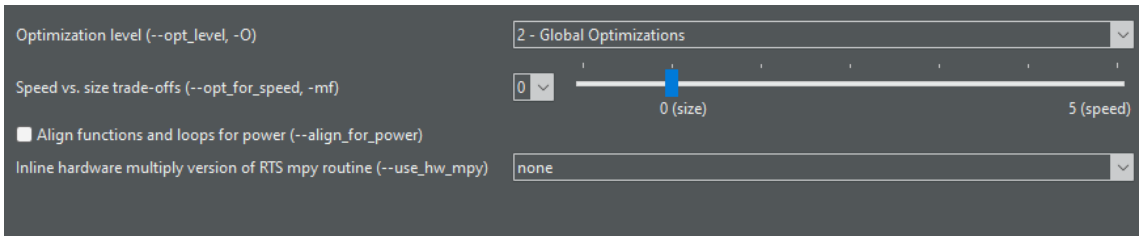
-actualiza_pantalla() se encarga de dibujar o atualizar los datos en pantalla que cambien en cada medición. Para ello recurre a funciones previamente mencionadas.

-settings() es la responsable de llamar a control_sel(), la cuál comprueba los valores de las banderas down,up y okey para decidir si hay que alternar alguna configuración o seleccionar otra opción. Por spuesto, si esto ocure, se guardará con la función guarda_flash() los nuevos valores para poderlos leer al reiniciar el micro.

-buzzer() realiza un barrido de rangos de distancias, para en función de ello elegir el DC de la señal PWM que recibirá el buzzer. Además, para conseguir esa intensidad e intermitencia buscada, se alternará entre PWM y E/S (buzzer apagado) conforme a la variable de tiempo t, de tal manera que a mayor distancia menor tiempo activo y con un periodo de intermitencia mayor.

Por último, comentar que para una mayor optimización se han elegido las variables volátiles cuidadosamente, de forma que el compilador pueda optimizar todas menos las que pueden ver alteradas su valor sin depender de nuestro código (por ejemplo, interrupciones).

Además, se ha decidido compilar el programa con la configuración:



De esta forma, se prioriza ocupar menos memoria frente al tiempo de compilación, que no nos es relevante para este caso, y se tiene capacidad para ampliar el proyecto si se desea. Además, permitimos al compilador optimizaciones globales para que consiga un mejor resultado (pero sin comprometer la funcionalidad del programa).

Resultados obtenidos

Se adjunta video de funcionamiento del sistema junto a la memoria del proyecto.

Código principal comentado

```
#include <msp430.h>
#include "grlib.h"
#include "Crystalfontz128x128_ST7735.h"
#include "HAL_MSP430G2_Crystalfontz128x128_ST7735.h"
#include <stdio.h>
#include "uart_STDIO.h"

//-----------PROTOTIPOS FUNCIONES-------------
void Set_Clk(void); //configura el reloj
void inicia_perifericos(void); //inicialización de periféricos
void primera_medida(void); //medida automática de distancia posterior al
conectar el móvil por Bluetooth por primera vez

void uart_putc(unsigned char c); //muestra caracter por terminal
void uart_puts(const char *str); //muestra frase por terminal
void uart_putfr(const char *str); //muestra frase y salto de línea por
terminal

void buzzer(void); //control del buzzer en función de la distancia_cm
void bluetooth(void); //control de transmisión de datos vía bluetooh
    void control_timers(bool sel,bool encendido); //gestiona el encendido o
apagado de los timers y la selección de que señal echo recibir (sensor
delantero o trasero)
void pantalla_espera(void); //pantalla de espera simple hasta que el usuario
se conecte por Bluetooth o envíe alguna señal si ya está conectado
void main_screen(void); //pantalla principal
    void pinta_fondo(void); //pinta fondo de la interfaz
    void pinta_titulo(void); //pinta el título del modo principal
    void pinta_simbolo_sonido(void); //decide si tachar o no el icono de
sonido
        void dibuja_altavoz(void); //dibuja icono de sonido
    void actualiza_pantalla(void); //dibuja todo lo que se debe actualizar en
función de la medida realizada
        void borra_barras(char numero); //se encarga de borrar las barras
sobrantes
        void dibuja_barras(char numero); //se encarga de dibujar las barras
faltantes
void settings(void); //gestiona la configuración del sistema
    void control_sel(void); //actualiza los valores de las variables de
configuración y de posición del indicador
    void pantalla_settings(void); //actualiza la pantalla de configuración
void guarda_flash(bool volume,bool idioma, bool unidades, bool interface);
//se encarga de guardar en flash la configuración actual para mantenerla al
inicio
void carga_config(void); //lee la flash para asignar sus valores a la
configuración al iniciar el programa
//--------------------------------------------

//------------VARIABLES GLOBALES---------------
Graphics_Context g_sContext;
volatile unsigned int distancia_cm; //unidades métricas
volatile unsigned int distancia_plg; //unidades imperiales
//---------------------------------------------------------
char cad_dist[15]; //cadena para mostrar el número de la distancia
volatile char t=0; //contador que incrementa su valor cada 65 ms | se utiliza
para gestionar la intermitencia del buzzer
```

```
volatile bool FinRx=0; //Fun de recepción de datos
volatile char comando_bluetooth='P'; //orden recibida por bluetooth
//-------------------------------------------------------
bool volume=1; //variable para activar o desactivar pitido de buzzer
bool volume_anterior=0; //variable para revisar si se ha modificado la
configuración de sonido
bool unidades=1; //1:Unidades métricas | 0:Unidades imperiales
bool idioma=1; //1:English | 0:Spanish
bool interface=1; //1:Dark | 0:Light
bool cambio_display=1; //sirve para saber si se debe actualizar o no la
pantalla
bool setting=0; //indica si estamos en el menú de configuración
bool up=0,down=0,okey=0; //flags para indicar si se ha recibido orden de
subida, bajada o cambio de valor respectivamente
char borrado; //número de barras a borrar en función de la distancia
calculada
char dibujado; //número de barras a dibujar en función de la distancia
calculada
char modo=3; //Modo: P=Parado M=Marcha R=Retroceso
unsigned long COLOR=0; //color principal de interfaz
unsigned long COLOR_inv=0x00FFFFFF; //color secundario de interfaz
char opt_sel=1; //indica cuál de las 4 opciones está elegida
char centro=35; //centro del cuadrado indicador de opción elegida
//-----------------------------------------------
void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
    inicia_perifericos();
    carga_config();
    pantalla_espera();
    LPM0; //se mantiene en la pantalla de espera hasta ser despertado por
interrupción al recibir conexión bluetooh de usuario o alguna orden
    primera_medida();
    comando_bluetooth='P'; //se considera que la conexión se hace en parado
    main_screen();
      while(1){
          LPM0; //esperar hasta recibir orden o realizar una medición
          bluetooth(); //modos de funcionamiento mediante comandos vía
bluetooth
          if(!setting) main_screen();
          else settings();
        buzzer(); //actualizar señal de buzzer
    }
}

//----------------------------------

//------RUTINAS DE INTERRUPCION----------
#pragma vector = TIMER1_A1_VECTOR //interrupción en los flancos de subida o
bajada del echo
__interrupt void EchoHCSR04(void){
        static char i=0; //variable auxiliar en interrupciones por flancos de
la señal echo
        static int tempo[2]; //vector para guardar en qué tiempos se producen
los flancos de subida y bajada del echo
        int diff; //tiempo que la señal echo permanece a 1
        tempo[i] = TA1CCR1; //copiar el valor del timer del registro TA1CCR1
```

```
        i += 1;
        TA1CCTL1 &= ~CCIFG ; //desactivar bandera de interrupción que se ha
activado para permitir nuevas interrupciones
        if (i==2) { //cuando se reciba tanto flanco de subida como de bajada
se calcula el tiempo
            diff=abs(tempo[i-1]-tempo[i-2]); //duracion señal echo en
microsegundos
            distancia_cm=diff/58; //Formula Datasheet: uS / 58 = centimeter
            distancia_plg=diff/148; //Formula Datasheet: uS / 148 = inch
            i=0;
            LPM0_EXIT; //despertar al tener la medida
        }
}

#pragma vector=TIMER1_A0_VECTOR //interrupción para contar cada 65ms
__interrupt void Contador_Buzzer(void)
{
    t++;
}

#pragma vector=USCIAB0RX_VECTOR
__interrupt void Recepcion_Orden_Bluetooh(void)
{
    //while (!(IFG2&UCA0TXIFG)); // Espera Tx libre
    UCA0TXBUF = UCA0RXBUF; // manda dato
    comando_bluetooth=UCA0RXBUF; //guardar valor recibido en esta variable
    FinRx=1; //indicar que se ha producido el fin de una recepción
    LPM0_EXIT; //despertar al recibir una orden
}

//-------------FUNCIONES--------------

void Set_Clk(void){
    BCSCTL2 = SELM_0 | DIVM_0 | DIVS_0;
    if (CALBC1_8MHZ != 0xFF) {
        __delay_cycles(100000);
        DCOCTL = 0x00;
        BCSCTL1 = CALBC1_8MHZ;      /* Set DCO to 8MHz */
        DCOCTL = CALDCO_8MHZ;
    }
    BCSCTL1 |= XT2OFF | DIVA_0;
    BCSCTL3 = XT2S_0 | LFXT1S_2 | XCAP_1;
}

void inicia_perifericos(void)
{
    Set_Clk();
    //----------PINES----------
    /*Pines de E/S */
    P1IES = 0;
    P1IFG = 0;
    P2OUT = 0;
    P2DIR = 0;
    P2IES = 0;
    P2IFG = 0;
```

```
    //Configuración de la USCI-A para modo UART:
    P1SEL2 |= BIT1 | BIT2;  //P1.1 RX, P1.2: TX
    P1SEL |= BIT1 | BIT2;
    P1DIR |= BIT2;

    //P1.6 TRIGGER (de ambos modulos ultranosido) (SALIDA) --> uso timer 0
comparador 1
    P1DIR |= BIT6; //P1.6 OUT
    P1SEL |= BIT6;  //MODO PWM
    P1SEL2 &= ~BIT6;

    //P2.1 ECHO (de un modulo ultrasonido) (ENTRADA) --> uso timer 1
comparador 1
    P2DIR &=~ BIT1; // P2.1 PWM
    P2SEL &=~ BIT1; //MODO I/O
    P2SEL2 &= ~ BIT1;

    //P2.2 ECHO (del otro modulo ultrasonido) (ENTRADA) --> uso timer 1
comparador 1
    P2DIR &=~BIT2; // P2.2 IN
    P2SEL &=~BIT2; //MODO I/O
    P2SEL2 &= ~BIT2;

    //PIN 2.4 --> TIMER 1 COMPARADOR 2 (MANEJO DEL BUZZER)
    P2DIR |= BIT4; //P2.4 OUT
    P2SEL &= ~ BIT4;  //MODO I/O (salida a 0)
    P2SEL2 &= ~BIT4;
    P2OUT &= ~BIT4;

    //--------TIMERS-------------
    //TIMER PARA TRIGGER
    TA0CTL=TASSEL_2|ID_3|MC_0;      //SMCLK, DIV=8 (1 MHZ) ,Up to CCR0
(inicialmente STOP)
    TA0CCTL1=OUTMOD_7;      //OUTMOD=7 --> PWM activa a nivel alto
    TA0CCR0=65535;   //periodo=65536 (65 ms aprox)
    TA0CCR1=10;      //DC (10 us)
    //TA0CCTL0=CCIE; //habilitamos interrupción

    //TIMER PARA ECHOS
    TA1CTL = TASSEL_2|ID_3|MC_0 ; //SMCLK, DIV=8 (1 MHZ) , Continous up
(incialmente STOP)
    TA1CCTL1 = CAP | CCIE | CCIS_0 | CM_3 | SCS ; //Capture/Compare mode (for
getting the time values), interrupts enable, capture/compare input, capture
mode as rising edge and falling edge together, synchronizing

    //TIMER PARA PWM BUZZER (TIMER 1 COMPARADOR 2)
    TA1CCTL2=OUTMOD_7;      //OUTMOD=7 --> PWM activa a nivel alto
    TA1CCTL0=CCIE; //habilitamos interrupción

    //PANTALLA BOOSTERPACK
    Crystalfontz128x128_Init();
    Crystalfontz128x128_SetOrientation(LCD_ORIENTATION_UP);
    Graphics_initContext(&g_sContext, &g_sCrystalfontz128x128);
    Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_BLACK); //fondo
en negro
    Graphics_clearDisplay(&g_sContext);
    Graphics_setFont(&g_sContext, &g_sFontCm14); //fuente
```

```
    /*------ Configuración de la USCI-A para modo UART:----------*/

    UCA0CTL1 |= UCSWRST; // Reset
    UCA0CTL1 = UCSSEL_2 | UCSWRST;
    UCA0MCTL = UCBRF_0 | UCBRS_6;
    /* UCSSEL_2 : SMCLK (8MHz)*/

    UCA0BR0 = 65; //UCA0BR1 && UCA0BR0 = 00000011 01000001
    UCA0BR1 = 3; //8M/833~=9600
    UCA0CTL1 &= ~UCSWRST; /* Quita reset */

    IFG2 &= ~(UCA0RXIFG); /* Quita flag */
    IE2 |= UCA0RXIE; /* y habilita int. */

    //FLASH
    FCTL2 = FWKEY + FSSEL_2 + 24; // 24 dividir frecuencia (dar tiempo para
correcta lectura memoria flash)

    __bis_SR_register(GIE);
}

void primera_medida(void){
    control_timers(0, 1); //seleccionamos sensor delantero y lo encendemos
    t=0;
    while(t<3); //espera hasta que se realice una medida
    control_timers(0, 0); //apagamos el sensor y el timer asociado a este
}

void buzzer(void){
    const unsigned int
modulo[8]={65535,50000,30000,20000,10000,5000,2000,1000}; //Duty Cycle(DC)
de señal PWM para el buzzer (modfica volumen)
    if(volume && modo!=3){ //si el volumen está activo distinguir rangos de
distancia para modular PWM
        //ZONAS POSIBLES
        if(distancia_cm<4){
            TA1CCR2=modulo[0];
            if(t>=1){
                t=0;
                P2SEL ^= BIT4;  //MODO PWM ó MODO I/O
            }
        }
        else if(distancia_cm<6 && distancia_cm>=4){
            TA1CCR2=modulo[1];
            if(t>=3){
                t=0;
                P2SEL ^= BIT4;  //MODO PWM ó MODO I/O
            }
        }
        else if(distancia_cm<8 && distancia_cm>=6){
            TA1CCR2=modulo[2];
            if(t>=6){
                t=0;
                P2SEL ^= BIT4;  //MODO PWM ó MODO I/O
            }
        }
        else if(distancia_cm<10 && distancia_cm>=8){
```

```
            TA1CCR2=modulo[3];
            if(t>=10){
                t=0;
                P2SEL ^= BIT4;  //MODO PWM ó MODO I/O
            }
        }
        else if(distancia_cm<15 && distancia_cm>=10){
            TA1CCR2=modulo[4];
            if(t>=20){
                t=0;
                P2SEL &= ~ BIT4; //MODO I/O
            }
            else if(t>=15 && t<20){
                P2SEL |= BIT4;  //MODO PWM
            }
            else P2SEL &= ~ BIT4; //MODO I/O
        }
        else if(distancia_cm<20 && distancia_cm>=15){
            TA1CCR2=modulo[5];
            if(t>=30){
                t=0;
                P2SEL &= ~ BIT4; //MODO I/O
            }
            else if(t>=25 && t<30){
                P2SEL |= BIT4;  //MODO PWM
            }
            else P2SEL &= ~ BIT4; //MODO I/O
        }
        else if(distancia_cm<25 && distancia_cm>=20){
            TA1CCR2=modulo[6];
            if(t>=40){
                t=0;
                P2SEL &= ~ BIT4; //MODO I/O
            }
            else if(t>=35 && t<40){
                P2SEL |= BIT4;  //MODO PWM
            }
            else P2SEL &= ~ BIT4; //MODO I/O
        }
        else if(distancia_cm<=30 && distancia_cm>=25){
            TA1CCR2=modulo[7];
            if(t>=50){
                t=0;
                P2SEL &= ~ BIT4; //MODO I/O
            }
            else if(t>=45 && t<50){
                P2SEL |= BIT4;  //MODO PWM
            }
            else P2SEL &= ~ BIT4; //MODO I/O
        }
        else{ //No buzzer --> lejos de objeto (distancia_cm>30 cm)
            P2SEL &= ~ BIT4;  //MODO I/O (salida a 0)
            P2SEL2 &= ~BIT4;
            P2OUT &= ~BIT4;
        }
    }
    else { //si el sonido no está activo
        //Desactivar PWM del buzzer
```

```
        P2SEL &= ~ BIT4;  //MODO I/O (salida a 0)
        P2SEL2 &= ~BIT4;
        P2OUT &= ~BIT4;
    }
}
void bluetooth(void){
        if(FinRx==1){ //si se ha dado el fin de una recepción de dato
            FinRx=0;
            switch(comando_bluetooth){ //distinguimos la orden recibida y
actuamos en función de ello
            case 'M': //marcha
                if(!setting){
                    uart_putfr("Modo marcha iniciado");
                    control_timers(0,1); //habilita sensor delantero
                    modo=1;
                }
                break;
            case 'R': //retroceso
                if(!setting){
                    uart_putfr("Modo retroceso iniciado");
                    control_timers(1,1); //habilita sensor trasero
                    modo=2;
                }
                break;
            case 'P': //parada
                if(!setting){
                    uart_putfr("Modo parada iniciado");
                    control_timers(0,0); //deshabilita sensores
                    modo=3;
                }
                break;
            /*-------------CONTROL CON MANDO---------------*/
            case 'O': //okey
                if(setting) //aseguramos que el usuario esté parado para
permitirle acceder al menú (para evitar distracciones)
                    okey=1;
                break;
            case 'U': //up
                if(setting) //aseguramos que el usuario esté parado para
permitirle acceder al menú (para evitar distracciones)
                    up=1;
                break;
            case 'D': //down
                if(setting) //aseguramos que el usuario esté parado para
permitirle acceder al menú (para evitar distracciones)
                    down=1;
                break;
            case 'S': //menú de opciones
                uart_puts("\n");
                if(modo==3){ //solo permitimos al usuario usar el menú de
configuración si está parado ( para evitar distracciones al volante)
                setting=!setting;
                cambio_display=1;
                }
                break;
        }
    }
}
```

```
void control_timers(bool sel,bool encendido){
    if(encendido){ //si se quiere usar un sensor (encendido=1) se encienden
los timers
        TA0CTL=TASSEL_2|ID_3|MC_1;      //SMCLK, DIV=8 (1 MHZ) ,Up to CCR0
        TA1CTL = TASSEL_2|ID_3|MC_2 ; //SMCLK, DIV=8 (1 MHZ) , Continous up
        switch(sel){ //para elegir sensor trasero o delantero
        case 0: //delantero
            TA1CCTL1 = CAP | CCIE | CCIS_0 | CM_3 | SCS ;
            P2SEL |= BIT1; //MODO PWM
            P2SEL &=~BIT2; //MODO I/O
            break;
        case 1: //trasero
            TA1CCTL1 = CAP | CCIE | CCIS_1 | CM_3 | SCS ;
            P2SEL &=~ BIT1; //MODO I/O
            P2SEL |= BIT2; //MODO PWM
            break;
        }
    }
    else { //si encendido=0 se apagan los timers y se ponen los pines a E/S
normal
        P2SEL &=~ BIT1; //MODO I/O
        P2SEL &=~BIT2; //MODO I/O
        TA0CTL=TASSEL_2|ID_3|MC_0;      //desactivamos timer TRIGGER
        TA1CTL = TASSEL_2|ID_3|MC_0 ; //desactivamos timer ECHO
    }
}

void pantalla_espera(void){
    Graphics_clearDisplay(&g_sContext);
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
    Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_CYAN);
    if(!idioma){
        Graphics_drawStringCentered(&g_sContext, "  Esperando  ", 14, 64, 40,
OPAQUE_TEXT);
        Graphics_drawStringCentered(&g_sContext, "  conexion   ", 14, 64, 55,
OPAQUE_TEXT);
        Graphics_drawStringCentered(&g_sContext, "por Bluetooh ", 14, 64, 70,
OPAQUE_TEXT);
    }
    else {
        Graphics_drawStringCentered(&g_sContext, "Waiting for ", 14, 64, 40,
OPAQUE_TEXT);
        Graphics_drawStringCentered(&g_sContext, " connection ", 14, 64, 55,
OPAQUE_TEXT);
        Graphics_drawStringCentered(&g_sContext, "via Bluetooh", 14, 64, 70,
OPAQUE_TEXT);
    }

}

void main_screen(void){
    if(cambio_display){
        comando_bluetooth='P'; //si se da un cambio de pantalla se asume que
el usuario está parado
        pinta_fondo();
        pinta_titulo();
    }
    pinta_simbolo_sonido();
```

```
    actualiza_pantalla();
    cambio_display=0;
}

void pinta_fondo(void){
    //---------FONDO----------------------------
    Graphics_setBackgroundColor(&g_sContext, COLOR_inv); //fondo en blanco
    Graphics_clearDisplay(&g_sContext);
    Graphics_setForegroundColor(&g_sContext, COLOR); // lo próximo se pintará
negro
    Graphics_Rectangle rect={0,25,127,115};
    Graphics_fillRectangle(&g_sContext,&rect); //rectángulo en franja central
    Graphics_setForegroundColor(&g_sContext, COLOR_inv);
    char ancho=60,altura;
    for(altura=25;altura<=45;altura++){
        Graphics_drawLineH(&g_sContext, 0, ancho, altura);
        ancho-=3;
    }
    ancho=60;
    for(altura=115;altura>=95;altura--){ //dibuja los triángulos
        Graphics_drawLineH(&g_sContext, 0, ancho, altura);
        ancho-=3;
    }
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
//remarcamos los bordes de rojo
    Graphics_drawLineH(&g_sContext, 60, 127, 25);
    Graphics_drawLineH(&g_sContext, 60, 127, 24);
    Graphics_drawLineH(&g_sContext, 60, 127, 115);
    Graphics_drawLineH(&g_sContext, 60, 127, 116);
    Graphics_drawLine(&g_sContext, 60, 24, 0, 44);
    Graphics_drawLine(&g_sContext, 60, 25, 0, 45);
    Graphics_drawLine(&g_sContext, 60, 116, 0, 96);
    Graphics_drawLine(&g_sContext, 60, 115, 0, 95);
}
void pinta_titulo(void){
    //--------------TITULO----------------------------------------
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLUE);
    if(idioma)
    Graphics_drawStringCentered(&g_sContext,"Parking System", 15, 64, 8,
TRANSPARENT_TEXT);
    else Graphics_drawStringCentered(&g_sContext,"Sistema Parking", 16, 64,
8, TRANSPARENT_TEXT);
    Graphics_drawLineH(&g_sContext, 12, 115, 18);
}
void pinta_simbolo_sonido(void){
    //--------------SIMBOLO SONIDO-------------------------------
    if((volume!=volume_anterior) || cambio_display){
        volume_anterior=volume;
        if(!volume){
            dibuja_altavoz();
            Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
            Graphics_drawLine(&g_sContext, 5, 119, 21, 106);
        }
        else{
            dibuja_altavoz();
        }
    }
}
```

```
void dibuja_altavoz(void){
    Graphics_setForegroundColor(&g_sContext, COLOR_inv);
    Graphics_Rectangle limpia={5, 106, 21, 119};
    Graphics_fillRectangle(&g_sContext,&limpia);
    Graphics_setForegroundColor(&g_sContext, COLOR);
    Graphics_Rectangle rect1={5,110,11,116};
    Graphics_drawRectangle(&g_sContext,&rect1);
    Graphics_drawLine(&g_sContext, 11, 110, 17, 106);
    Graphics_drawLine(&g_sContext, 11, 116, 17, 119);
    Graphics_drawLineV(&g_sContext, 17, 106, 119);
    Graphics_drawLineV(&g_sContext, 19, 110, 116);
    Graphics_drawLineV(&g_sContext, 21, 108, 118);
}

void actualiza_pantalla(void){
    //--------------dinámica de pantalla--------------
    Graphics_setForegroundColor(&g_sContext, COLOR);
    switch(modo){
    case 1:
        if(!idioma)
            Graphics_drawString(&g_sContext, " Marcha  ", 10, 63, 90,
OPAQUE_TEXT);
        else Graphics_drawString(&g_sContext, " Forward ", 10, 63, 90,
OPAQUE_TEXT);
        break;
    case 2:
        if(!idioma)
            Graphics_drawString(&g_sContext, "Retroceso", 10, 63, 90,
OPAQUE_TEXT);
        else Graphics_drawString(&g_sContext, " Backward", 10, 63, 90,
OPAQUE_TEXT);
        break;
    case 3:
        if(!idioma)
            Graphics_drawString(&g_sContext, "  Parada  ", 10, 63, 90,
OPAQUE_TEXT);
        else Graphics_drawString(&g_sContext, "  Stop   ", 10, 63, 90,
OPAQUE_TEXT);
        break;
    }
    if(comando_bluetooth!='P' || cambio_display){
        if(distancia_cm<30)
            borrado=5-(5-distancia_cm/5); //cálculo de barras a borrar
        else borrado=6;
        borra_barras(borrado);
        dibujado=6-borrado; //cálculo de barras a dibujar
        dibuja_barras(dibujado);
        Graphics_setForegroundColor(&g_sContext, COLOR);
        if(unidades)
                sprintf(cad_dist,"  %d cm  ",distancia_cm);
        else {
            if(!idioma) sprintf(cad_dist,"  %d plg  ",distancia_plg);
            else sprintf(cad_dist,"  %d inch",distancia_plg);
        }
        Graphics_drawString(&g_sContext,cad_dist, 15, 65, 63, OPAQUE_TEXT);
        switch(dibujado){
        case 0:
```

```
            if(!idioma)
                Graphics_drawString(&g_sContext,"Muy lejos ", 11, 62, 35,
OPAQUE_TEXT);
            else Graphics_drawString(&g_sContext,"   So far    ", 11, 62, 35,
OPAQUE_TEXT);
            break;
        case 1:
        case 2:
            if(!idioma)
                Graphics_drawString(&g_sContext,"  Lejano  ", 11, 62, 35,
OPAQUE_TEXT);
            else Graphics_drawString(&g_sContext,"   Far    ", 11, 62, 35,
OPAQUE_TEXT);
            break;
        case 3:
        case 4:
            if(!idioma)
                Graphics_drawString(&g_sContext," Proximo  ", 11, 62, 35,
OPAQUE_TEXT);
            else Graphics_drawString(&g_sContext,"   Near   ", 11, 62, 35,
OPAQUE_TEXT);
            break;
        case 5:
        case 6:
            if(!idioma)
                Graphics_drawString(&g_sContext,"Muy cerca ", 11, 62, 35,
OPAQUE_TEXT);
            else Graphics_drawString(&g_sContext,"Very close", 11, 62, 35,
OPAQUE_TEXT);
            break;
        }
    }
}

void borra_barras(char numero){
    Graphics_setForegroundColor(&g_sContext, COLOR);
    char j=0,h=35;
    for(j=54;j>54-10*numero && j<200;j-=10){ //REVISAR EL >=
        Graphics_Rectangle borra_barra={j,70-h,j+4,70+h};
        Graphics_fillRectangle(&g_sContext, &borra_barra); //borra las barras
        h-=5;
    }
}

void dibuja_barras(char numero){
    char j=0,h=5,aux=1;
    for(j=4;j<4+10*numero;j+=10){
        switch(aux){
        case 1: Graphics_setForegroundColor(&g_sContext,
GRAPHICS_COLOR_GREEN); break;
        case 2: Graphics_setForegroundColor(&g_sContext,
GRAPHICS_COLOR_GREEN); break;
        case 3: Graphics_setForegroundColor(&g_sContext,
GRAPHICS_COLOR_YELLOW); break;
        case 4: Graphics_setForegroundColor(&g_sContext,
GRAPHICS_COLOR_YELLOW); break;
        case 5: Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
break;
```

```
        case 6: Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
break;
        }
        Graphics_Rectangle dibuja_barra={j,70-h,j+4,70+h}; //dibuja las
barras
        Graphics_fillRectangle(&g_sContext, &dibuja_barra);
        h+=5;
        aux++;
    }
}

void settings(){
    control_sel();
    if(interface){
        COLOR=0;
        COLOR_inv=0x00FFFFFF;
    }
    else{
        COLOR=0x00FFFFFF;
        COLOR_inv=0;
    }
    pantalla_settings();
}
void control_sel(void){
    //-------------CONTROL POR MANDO-----------------
    if(down){
        if(opt_sel==4){
            opt_sel=1;
            centro=35;
        }
        else {
            opt_sel++;
            centro+=25;
        }
        down=0;
    }
    else if(up){
        if(opt_sel==1){
            opt_sel=4;
            centro=110;
        }
        else{
            opt_sel--;
            centro-=25;
        }
    up=0;
    }
    if(okey){
        switch(opt_sel){
        case 1:
            volume=!volume;
            break;
        case 2:
            idioma=!idioma;
            break;
        case 3:
            unidades=!unidades;
            break;
```

```
        case 4:
            interface=!interface;
            break;
        }
        guarda_flash(volume,idioma,unidades,interface);
        okey=0;
    }
}

void pantalla_settings(void){
    //-----------Pantalla principal menú------------
    Graphics_setBackgroundColor(&g_sContext, COLOR); //fondo en blanco
    Graphics_clearDisplay(&g_sContext);
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_DARK_GREEN);
    Graphics_Rectangle rectan={8,centro-2,12,centro+2};
    Graphics_fillRectangle(&g_sContext, &rectan); //dibujar indicador de
opción
    if(idioma)
    Graphics_drawStringCentered(&g_sContext,"Settings", 10, 64, 7,
TRANSPARENT_TEXT);
    else Graphics_drawStringCentered(&g_sContext,"Configuración", 10, 64, 7,
TRANSPARENT_TEXT);


    Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_DARK_GREEN);
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_GREEN_YELLOW);
    if(idioma){
        Graphics_drawString(&g_sContext,"Volume:", 9, 20,30, OPAQUE_TEXT);
        Graphics_drawString(&g_sContext,"Language:", 10, 20,55, OPAQUE_TEXT);
        Graphics_drawString(&g_sContext,"Units:", 7,20, 80, OPAQUE_TEXT);
        Graphics_drawString(&g_sContext,"Interface:", 10,20, 105,
OPAQUE_TEXT);
    }
    else {
        Graphics_drawString(&g_sContext,"Volumen:", 9, 20,30, OPAQUE_TEXT);
        Graphics_drawString(&g_sContext,"Lenguaje:", 10, 20,55, OPAQUE_TEXT);
        Graphics_drawString(&g_sContext,"Unidades:", 10,20, 80, OPAQUE_TEXT);
        Graphics_drawString(&g_sContext,"Interfaz:", 10,20, 105,
OPAQUE_TEXT);
    }

    //----------opciones--------
    if(volume)
        Graphics_drawString(&g_sContext,"ON", 5,80, 30, OPAQUE_TEXT);
    else Graphics_drawString(&g_sContext,"OFF", 5,80, 30, OPAQUE_TEXT);

    if(idioma)
        Graphics_drawString(&g_sContext,"ENG", 5,90, 55,OPAQUE_TEXT);
    else Graphics_drawString(&g_sContext,"ESP", 5,90, 55,OPAQUE_TEXT);

    if(unidades)
        Graphics_drawString(&g_sContext,"cm", 5,85, 80,OPAQUE_TEXT);
    else {
        if(idioma)
            Graphics_drawString(&g_sContext,"inch", 5,85, 80,OPAQUE_TEXT);
        else Graphics_drawString(&g_sContext,"plg", 5,85, 80,OPAQUE_TEXT);
    }
    if(interface){
```

```
        if(idioma)
            Graphics_drawString(&g_sContext,"Dark",8,85, 105,OPAQUE_TEXT);
        else Graphics_drawString(&g_sContext,"Oscuro",10,85,
105,OPAQUE_TEXT);
    }
    else{
        if(idioma)
            Graphics_drawString(&g_sContext,"Light",8,85, 105,OPAQUE_TEXT);
        else Graphics_drawString(&g_sContext,"Claro",8,85, 105,OPAQUE_TEXT);
    }

}

void guarda_flash(bool volume,bool idioma, bool unidades, bool interface){

    char *  Puntero = (char *) 0x1000; // Apunta a la direccion
    char copia_config[4]={volume,idioma,unidades,interface};
    char n;
        //Borra bloque
        FCTL1 = FWKEY + ERASE;       // activa  Erase
        FCTL3 = FWKEY;               // Borra Lock (pone a 0)
        *Puntero = 0;             // Escribe algo para borrar el segmento

        //FCTL3 = FWKEY;               // Borra Lock (pone a 0)
        FCTL1 = FWKEY + WRT;         // Activa WRT
        for(n=0;n<4;n++) Puntero[n]=copia_config[n]; // Escribe la
configuración en 0x1000+n
        FCTL1 = FWKEY;               // Borra bit WRT
        FCTL3 = FWKEY + LOCK;        //activa LOCK
}

void carga_config(void){
    //Guarda variables en RAM
    char k;
    char *  Puntero = (char *) 0x1000; // Apunta a la direccion

    for (k=0;k<4;k++)
        switch(k){
        case 0:
            volume=Puntero[k];
            break;
        case 1:
            idioma=Puntero[k];
            break;
        case 2:
            unidades=Puntero[k];
            break;
        case 3:
            interface=Puntero[k];
            break;
        }
}
void uart_putc(unsigned char c)
{
    while (!(IFG2&UCA0TXIFG)); // Espera Tx libre
    UCA0TXBUF = c;             // manda dato
}
```

```
void uart_puts(const char *str)
{
     while(*str) uart_putc(*str++); //repite mientras !=0
}

void uart_putfr(const char *str)
{
     while(*str) uart_putc(*str++);
     uart_puts("\n");     //termina con CR
}
```