PROYECTO FINAL MICROCONTROLADORES*

Diseño e implementación de sistema electrónico de apoyo para personas ciegas o con debilidad visual

Rioja Cintya, García Alan y Ruiz Pablo.
IMD
ITESM CCM
Ciudad de México
A01214551@itesm.mx A01214651@itesm.mx A13@itesm.mx

***Abstracto*—El siguiente documento presenta el desarrollo de un sistema basado en un microcontrolador ATMEGA16 que funciona como auxiliar en el movimiento de personas ciegas o con debilidad visual en distintos grados a través del trabajo combinado de un sensor ultrasónico, un servomotor, una LCD y un sistema mínimo para el microcontrolador. El reporte incluye una breve introducción del contexto y las circunstancias bajo las que se desarrolla el dispositivo, una descripción general de los requerimientos de hardware y software implementados y una conclusión respecto a los resultados del proyecto.**

Palabras clave—ATMEGA16, ciego, débil visual, sensor ultrasónico, servomotor.

I. Planteamiento del problema

Se desea implementar un sistema basado en ATMEGA16 que sea capaz de controlar y coordinar el funcionamiento de un servomotor y un sensor ultrasónico tanto para la detección de objetos circundantes al sensor, como para la medición de distancia entre éste y los objetos detectados. El sistema deberá trabajar en cuatro modos de operación diferentes, uno diseñado para el perfil de un usuario con ceguera, dos para un usuario con debilidad visual y el último para un usuario en general. El sistema debe ser capaz de mostrar un menú con las opciones de trabajo además de contar con un teclado que permita la selección de éstas.

En el caso del modo diseñado para personas con ceguera, cuando el sistema detecte la presencia de un objeto a menos de un radio de 1 metro del sensor será necesario que la medición de proximidad se indique únicamente a través de la modulación del sonido de tres buzzers.

En el caso de los modos diseñados para personas con debilidad visual, cuando el sistema detecte la presencia de un objeto a menos de un radio de 1 metro del sensor será necesario que la medición de proximidad se indique de forma visual en un mensaje que contenga o únicamente la distancia medida o ésta en conjunto con la dirección donde se hizo la detección. En el primer caso, es necesario que la indicación de dirección se realice de forma auditiva a través de tres buzzers.

En el caso del último modo, el sistema debe medir y mostrar de forma visual únicamente la distancia entre el sensor y un objeto cercano en una sola dirección, la cual puede establecerse en tres opciones por medio del movimiento de un joystick. Además, el conjunto de datos adquiridos por el sistema debe ser enviado por medio de comunicación serial a una computadora que se encuentre conectada al sistema.

II. Modos de operación del sistema

Modo 1. Ceguera: Activado a través de la tecla 1.

Modo 2. Débil visual 1: Activado a través de tecla 2. En este modo la detección de objetos se indica de forma visual y sonora.

Modo 3. Débil visual 2: Activado a través de tecla 3. La detección de objetos se indica únicamente de forma visual.

Modo 4. Comando prueba: Activado por medio de tecla.

El sensor detecta objetos en una dirección fija, la cual puede seleccionarse en tres opciones a través del movimiento del joystick en un solo eje.

III. Requerimientos de hardware

Se utilizó como microcontrolador al modelo ATMEGA16, el cual tiene cuatro puertos de entrada y salida de 8 bits; cada pin tiene una función secundaria especial que se relaciona con los bloques funcionales especiales que integran al microcontrolador.

Para este proyecto el microcontrolador contó con un sistema mínimo conformado por un circuito de reinicio (un resistor, un capacitor y un botón) y una fuente de alimentación de 5 V. La fuente de la frecuencia de funcionamiento del microcontrolador se decidió que fuera el oscilador RC interno de 8 MHz. Por otra parte, se utilizaron tres puertos de entrada/salida del microcontrolador, el A, B y D, y se utilizó la unidad de captura, los tres temporizadores, el convertidor analógico digital, el módulo USART y el módulo de interrupciones externas y por temporizador que integran al microcontrolador (Fig. 1).

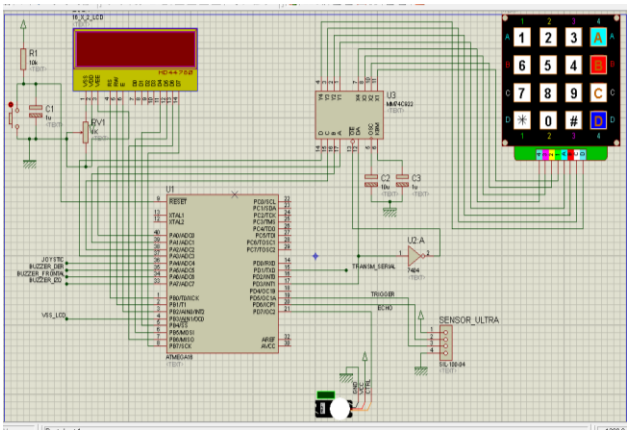

Fig. 1. Microcontrolador ATMEGA16 y los diferentes periféricos de entrada utilizados en el proyecto.

Como elementos fundamentales del proyecto se utilizaron un servomotor, el cual controla su desplazamiento a partir de una señal con PWM, y un sensor ultrasónico que funciona a partir del envío hacia éste de una señal de activación, un pulso en alto de 10 µs, y la generación de una señal con ancho de pulso variable como respuesta a la detección de un eco y que es proporcional a la distancia existente entre el sensor y el objeto que haya producido el eco.

Relacionando los elementos fundamentales del sistema con el microcontrolador (Fig. 2), el servomotor fue controlado a partir del pin D7 u OC2, el cual funciona como puerto de salida de la señal generada por el temporizador 2. La señal de activación del sensor ultrasónico, 10 µs de duración en estado alto, fue generada por software y entregada al sensor a través del pin D5, mientras que la señal de salida del sensor, señal de ancho de pulso variable, fue entregada para su análisis al microcontrolador a través del pin D6 o ICP1, el cual corresponde al puerto de entrada de la señal leída por la unidad de captura del microcontrolador.

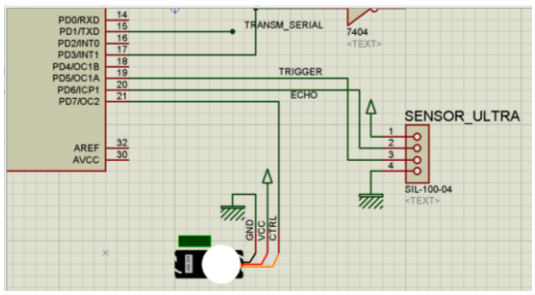

Fig. 2. Relación entre servomotor y sensor ultrasónico con microcontrolador.

La visualización del menú de opciones, además de la distancia medida y/o la dirección desde la cual se midió ésa, fue hecha con el uso de una pantalla de cristal líquido (LCD) de 16 caracteres de 5x10 puntos máximo y dos líneas para escritura. Como características particulares, se menciona que la pantalla se configuró para utilizarse con un canal de datos de 4 bits, con caracteres de 5x8 puntos, con habilitación de escritura en las dos líneas, con el cursor deshabilitado y que se utilizó un potenciómetro de 1 kΩ para regular la intensidad de brilla de la pantalla. Respecto a la interacción de la LCD con el microcontrolador (Fig. 3), todo el puerto B de éste fue dedicado para el control de la pantalla. Particularmente, los pines B0, B1 y B2 se encargaron del estado de las señales de control RS, RW y Enable respectivamente, B3 se encargó del estado del pin VS y los pines B4, B5, B6 y B7 funcionaron como el canal de datos de 4 bits para la LCD.

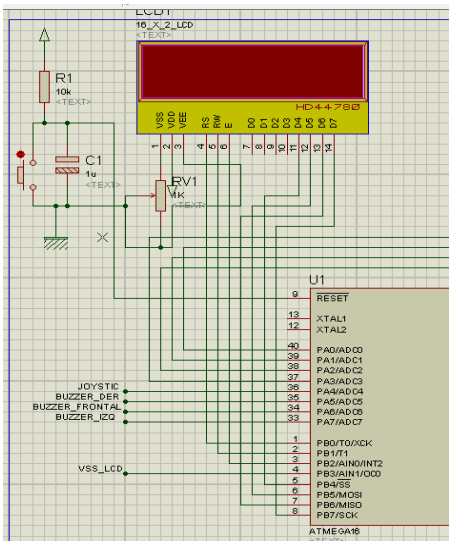

Fig. 2. Relación entre LCD con microcontrolador.

La modulación de la frecuencia de activación de los buzzers fue realizada por medio del temporizador 1 del ATMEGA16. Para esto fue necesario emplear los tres últimos pines del puerto A del microcontrolador, A5, A6 y A7, como puertos de salida de señales de frecuencia variable y ciclo de trabajo constante (Fig. 4). Éstas no se conectaron directamente a los buzzers, sino que fueron dirigidas a la base de un transistor NPN encargado de brindar la corriente necesaria para la activación óptima del buzzer (Fig. 5).

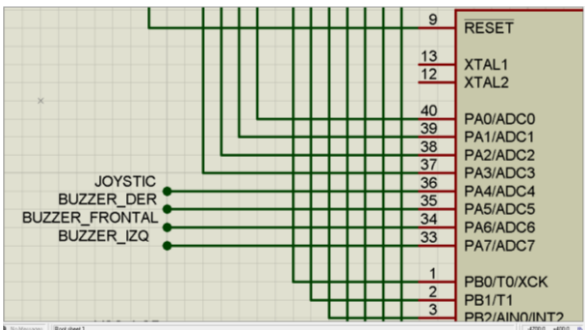

Fig. 3. Relación entre periféricos y puerto A del microcontrolador.

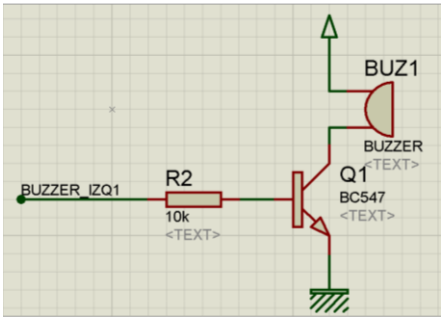

Fig. 4. Configuración para activación de buzzer mediante señal de microcontrolador con transistor NPN.

El sistema también utilizó un teclado matricial 4x4 para la selección de los disintos modos de operación del sistema, así como para su encendido y apagado. Para el control del teclado matricial se utilizó un circuito M74C922 (Fig. 5). Éste conecta a un circuito pull-up interno a todas las columnas del teclado matricial y únicamente se encarga de generar un barrido a alta frecuencia de señales en estado bajo en las filas del teclado. Entonces, la detección de la presión de una tecla se realiza a partir de identificar dos estados bajos simultáneos en alguna fila y en alguna columna. Por ejemplo, la tecla 1 se identifica a partir de que la señal de la columna 1 (X1) y la señal de la fila 1 (Y1) están en estado bajo.

Una vez que se detectó la presión de alguna tecla, el circuito se encarga de relacionar el evento con algún código binario entre 0 y 15, el cual mostrará en su canal de salida de datos (de 4 bits). Antes de habilitarlo, éste se encuentra en el estado de alta impedancia y únicamente mostrará la información correspondiente cuando la señal del pin Output Enable (OE) se encuentre en estado bajo. Además de esta señal de control, el circuito M74C922 también genera una señal activa en estado alto denominada Available, la cual indica al usuario que la generación del código binario ha finalizado y éste puede mostrarse en el canal de datos.

Por lo tanto, la secuencia de eventos a seguir para leer un dato del teclado matricial es la siguiente: se debe presionar una tecla, luego se debe detectar el flanco de subida de la señal Available, evento que debe disparar la activación en estado bajo de la señal Output Enable. En este momento el código binario asociado a la tecla presionada se mostrará en el canal de salida de datos, por lo cual éste debe ser leído por algún puerto del microcontrolador.

Relacionando al microcontrolador y el circuito de control del teclado matricial, el nibble bajo del puerto A (A0, A1, A2 y A3) funciona como canal de entrada para el código BCD asociado a cada tecla y que es generado por el circuito 74C922. El pin D2 o INT1 funciona como puerto de entrada para la señal Available, la cual generará una interrupción externa como aviso de que se presionó una tecla y que es necesario leer el nibble bajo del puerto A del microcontrolador.

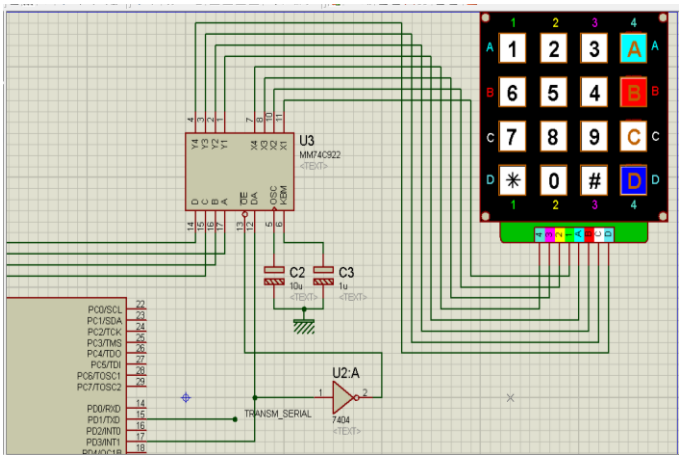

Fig. 5. Relación entre circuito de control de teclado matricial y microcontrolador.

Siguiendo con la descripción del puerto A del microcontrolador, el pin A4 se configuró como puerto de entrada para la señal analógica proveniente de un joystick y por lo tanto se activó su función especial secundaria como entrada 4 para el convertidor analógico-digital del microcontrolador (Fig. 3).

Finalmente, el uso de la unidad USART del microcontrolador para transmitir datos de forma serial de éste hacia una computadora necesitó la implementación de un circuito de acoplamiento para señales de los protocolos RS-232 y serial TTL. Para esto se utilizó un circuito MAX232 (Fig. 6), el cual se encargó de acondicionar los niveles de voltaje del protocolo RS232 con aquéllos manejados en los circuitos TTL. Las señales de origen de la computadora ingresan al MAX232 a traves de un conector USB-serial DB9, particularmente de los pines 2 y 3 de la terminal DB9. Las señales de entrada a los pines RXD (D0) y TXD (D1) del microcontrolador provendrán de las terminales 9 y 10, respectivamente, del circuito MAX232. En esta ocasión, al no realizarse ninguna recepción de datos por parte del microcontrolador, fue posible descartar el uso del pin D1 o TXD del microcontrolador.

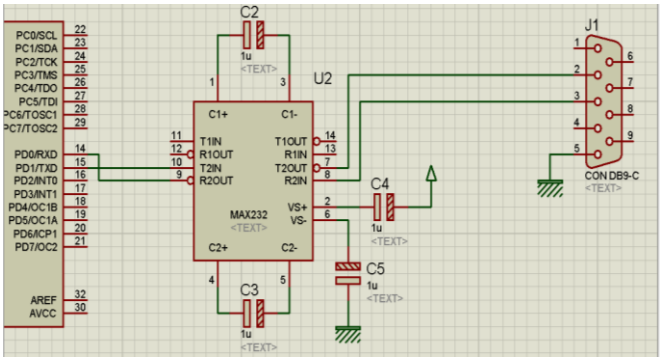

Fig. 6. Relación entre circuito de control de comunicación serial y el microcontrolador.

IV. Requerimientos de software

Dentro de los requerimientos de software se encuentran, la implementación de timers, de interrupciones y subrutinas. Se empezará por explicar los vectores de interrupciones con su descripción detallada.

Para poder tener las palabras de control de los timers, se procedió a la implementación de los modos para generar PWM, para el caso del timer 2 se empleó dicha modalidad con pre-escaler 1024.

Para generar las palabras de control necesarias se partió del hecho de que se necesitaba emplear una frecuencia de 300[Hz], que es lo mismo a decir que se necesitaba una frecuencia de 3.33 [ms], para esto es necesario generar un ancho de pulso del 72% o 2.4 [ms], 18% o 0.6 [ms] y 45% o 1.5 [ms] de dicho periodo para obtener las posiciones deseadas. Para este caso se propusieron tres posiciones; derecha, enfrente e izquierda.

Para poder obtener el valor del TCNT2 para la generación de la frecuencia se realzó el siguiente cálculo:

$$TCNT2 = \frac{3.33[ms]}{(1024) * (125 \times 10^{-9})} = 25 \qquad [1]$$

Como se aprecia en la ecuación 1, se requieren 25 pulsos para generar la frecuencia deseada, es por ello que se le carga un valor de 230 al TCNT2 para que empiece desde ese valor y al desbordarse se generen dicha frecuencia. Ahora bien con este valor, se procedió a hacer el cálculo con una relación lineal de este valor con los porcentajes previamente descritos. Al final los valores quedaron de la siguiente manera:

Neutral: 240 equivalente a un 45% de duty cycle.
Mínimo: 252 equivalente a un 18% de duty cylce.
Máximo: 238 equivalente a un 72% de duty cycle.

Dichos valores serán almacenados a partir de la dirección 0x60 de RAM. Para las palabras de control se emplearon los siguientes parámetros, en donde se especifica de manera detallada de acuerdo con los requerimientos del programa.

Palabras de control:

TCCR2: Palabra de control del timer 2; pre-escala de 1024, modo Fast PWM, modo no invertido.

TCCR0: Palabra de control del timer 0, pre-escala de 1024.

GICR: Palabra de control para las interrupciones externas en donde se habilita la interrupción externa 1.

MCUCR: Habilita la interrupción externa 1 con flanco de subida.

TIMSK: Palabra de control para la interrupción por timer, activa TMR1 UCP timer 1 por unidad de captura; TIMR2 Compare timer 2 por compare; TMR0 Compare interrupción de timer 0 por compare.

TCCR1A: Palabra de control para el timer 1 para que trabaje en modo normal.

TCCR1B: Palabra del timer 1 para que trabaje con un pre-escaler de 1024 en modo normal, que detecte el flanco de subida de la unidad de captura.

TCCR1A, 2: Palabra de control para el timer 1 para que se trabaje en modo CTC, genere una señal PWM con salida deshabilitada.

UCSRC: Palabra de registro de control para la comunicación serial, en modo asíncrono, sin paridad con 1 bit de parada con un tamaño de palabra de 8 bits.

UCSRB: Se habilita la transmisión serial.

UBRRL: Para un baudrate de 14 400 baudios.

ADMUX: Se la alimentación interna de 2.56 [V], y se activa el canal análogo 4.

ADCSRA: Se activa la habilitación del ADC. Con preescales de 128.

Ahora se procederá a describir las interrupciones empleadas:

- EXT1 0x04 **“Teclado”:** se entra a esta interrupción cada que se presiona una tecla del teclado matricial.
- TIMR2 overflow 0x08 **“Frecuencia”**: sirve para generar una frecuencia de 300[Hz] y poder mover el servomotor, se entra a esta interrupción cada que se desborda, se sobrepasa el 255.
- TIMR1 captura 0x0A **“tmr1c”**: empleada para la unidad de captura, se entra cada que se detecta un cambio de estado en el pin ICP.
- TIM1 Compare A, 0x0C **“tmr1ca”**: para generar frecuencias en los buzzers, se entra cada que hay una comparación o igualdad con el contador del timer 1.
- Timer 0 Compare 0x26 **“cambio”:** se entra a esta interrupción cada 10[ms] con la finalidad de generar retardos de 1 y 2 segundos respectivamente para la activación del sensor ultrasónico.

Siguiendo con la lógica del programa se procedió a habilitar los puertos que se emplearían. Se poseen 4 diferentes puertos dentro del microcontrolador, de los cuales para este proyecto solo se hizo uso de 3. El puerto A sirvió para poder tener entrada del teclado matricial (PA0-PA3), una entrada analógica (PA4) y el control de los buzzers (PA5-PA7). El puerto B fue destinado para el control de la LCD. Por último el puerto D se empleó para diferentes funciones, siendo estas:

- PD0: sin función.
- PD1: Transmisión serial. (Tx)
- PD2: sin función.
- PD3: entrada de la interrupción externa 1.
- PD4: sin función.
- PD5: salida del trigger que activa el sensor ultrasónico.
- PD6: entrada del pulso (echo) del sensor ultrasónico, entrada para la unidad de captura.
- PD7: OC2 salida PWM para el servomotor.

La siguiente sección es el main, para poder discernir entre las modalidades seleccionadas a partir del teclado matricial se designó un espacio en la memorai RAM 0x69 el cual posee el valor de la tecla presionada. Una vez obtenido este valor se procede a comparar dicho valor y direccionar la instrucción a la sección donde se realizará la acción previamente descrita de acuerdo con la modalidad seleccionad.

En general se comparte una subrutina para las diferentes modalidades y se va cambiando algunas características, tales como la activación de buzzers o de la LCD. Para ello es preciso emplear descartar el modo prueba ya que es el único que tiene una función diferente. Lo primero que se debe de hacer es limpiar la LCD y dar inicio a todos los timers, Se espera a que se hagan los tres cambios de posiciones del servo motor y vuelvo a cargar el puntero Y para la adquisición de nuevos datos obtenidos por medio de la unidad de captura. Posterior a esto llamo a la rutina para que dichos valores obtenidos por la unidad se puedan procesar y sacar el promedio, este nuevo valor es almacenado en la posición 0x6A, 0x6B y 0x6C de RAM respectivas a las posiciones de los servo motores. Ahora bien cuando se termina este proceso se vuelve a comparar el valor para saber si se empleará la LCD o no, es decir se compara con el valor de 2 y 3 ya que son las modalidades en donde se emplea el despliegue de la distancia. De ser este el caso se prepara la LCD junto con el puntero a ROM para que se visualice en la LCD la distancia y dependiendo del modo despegar la dirección de la misma.

Si se está en un modo en donde se activen los buzzers se carga el valor de la distancia obtenida, se prepara el timer 1 para la generación de la frecuencia y por último se llama a la subrutina para poder encender el buzzer deseado.

Cabe mencionar que en cada momento que se esté activando un buzzer se tiene que cargar el valor actual del servo en una dirección de RAM pre destinado.

En otra modalidad, denominada prueba se tiene que detener el servo motor y poder controlar la dirección del mismo mediante un joystick; aunado a esta función se debe de emplear la comunicación serial y desplegar dicha información en el programa terminal. Para lograr lo anterior se sigue un procedimiento parecido al anterior en el cual se vuele a comparar el valor almacenado en la dirección de memoria “tecla” con la finalidad de validad que en efecto se esté presionando la tecla 4.

Una vez validado este valor se borra la pantalla de la LCD y se apagan los timers, reinicio los contadores e inicio la primera conversión que se obtiene con el joystick. Este proceso se hace mediante la espera, es decir se hace un “polling”. Después de que se ha terminado la conversión se deshabilita la conversión por polling y se lee la palabra convertida en los registros ADCH y ADCL. Para esta sección se debe de mencionar que valores mayores a 900 indican una posición de 180° para el servomotor, valores menores a eso indican una posición de 0° del servo motor y valores iguales indican la posición neutral del mismo. Al poseer dos registros que indican la parte alta y la parte baja del número en cuestión se procede a comparar el ADCH con la parte alta del 900 y de ser iguales se pasa a otro proceso en donde se descarta que sean iguales; de lo contrario se indica que se está en la posición máxima almacenando el valor de OCR adecuado. Así se sigue el proceso hasta traducir el valor adquirido por el joystick a una posición del servo.

Una vez terminado este proceso se prosigue a preparar la unidad de captura para realizar las adquisiciones pertinentes; despliego los valores adquiridos en la LCD y posteriormente se prepara el sistema para enviar los valores mediante comunicación serial. Se propuso a enviar 32 valores adquiridos mediante polling por lo que se manda el primer valor al buzz de transmisión y se espera a que se encuentre vacío, esto se logra al preguntar el estado de una bandera la cual indica el fin de transmisión. Una vez terminado todo este proceso se vuelve a reiniciar el programa y preguntar por el valor de la tecla seleccionada.

Ahora se procederá a describir a detalle la función de cada una de las interrupciones:

Frecuencia: Se entra cada 3.33 [ms], en etsa sección se carga la posición actual del servo a la dirección 0x64 de RAM esto es con la finalidad de cambiar el 0CR2 y poder tener una señal PWM variable. Aunado a lo anterior se carga el contador del timer en 230 para generar frecuencias de 300 [Hz].

Teclado: Se entra a esta interrupción cada que se presiona una tecla del teclado matricial, una vez dentro de esta interrupción se lee el valor del puerto A, pero solo se desea conservar la parte baja de la palabra por lo que se le aplica una máscara y se guarda el valor de interés en la dirección 0x69 de RAM.

Tmr1ca: Se entra cada que existe una comparación con el timer 1 y se encuentra una igualdad con el OCR1AH y OCR1AL, una vez que esto sucede se cargan los valores de las señales con las frecuencias pertinente, aunado a eso se genera un toggle en el puerto del buzzer activo.

Cambio: Se entra a dicha interrupción cada 10 [ms], aunado a eso se posee un contador con un valor de 200 el cual se compara dos veces con la finalidad de saber cuándo ha pasado un segundo y poder activar el sensor ultrasónico. Aunado a esto se debe mencionar que se habilitan las interrupciones aquí adentro para no perder la señal PWM que sirve para mover el servomotor. Cuando ha pasado un segundo habilito el sensor ultrasónico, para ello se manda una señal en alto al pin PD5 durante 10[µs] y se apaga después de ese tiempo. La segunda vez que detecta que ha pasado un segundo se vuelve a activar el sensor ultrasónico, como el proceso anterior y posteriormente se vuelve a cargar el valor del contador para seguir con el proceso indefinidamente. Pasado esto se genera un retardo de 10[ms] para recibir el último valor entregado por el sensor. Aquí dentro checo el valor almacenado en “tecla” con la finalidad de descartar que se encuentre en el modo prueba, de ser igual el programa se sale de la interrupción de lo contrario se extrae el valor de la posición del servo motor que se encuentra en RAM y se carga al OCR2. Aunado a esto se posee un contador con un valor 3 que sirve para verificar que se han pasado las tres posiciones del servomotor.

Tmr1c: Entra en primera instancia cuando se detecta un cambio en alto en el pin PPD6 (ICP) que es el pin en donde se recibe la señal de echo proveniente del sensor ultrasónico. Una

vez que se detecte dicho cambio se cambia la sensibilidad del timer para que ahora se detecte una falling edge de la señal proveniente del sensor. Cada vez que se detecte lo anterior se guardaran los valores capturados de los timers TCNT1H y TCNT1L en los registros de la unidad de captura ICRH e ICRL. Posterior a esto se guardan todos los valores a partir de la dirección 100 de la RAM y se apaga la bandera de unidad de captura.

Para entender la lógica del sistema se procederá a describir las subrutinas empleadas dentro del programa.

Prepararnumero: Esta subrutina está encargada de preparar el número binario entregado por la unidad de captura, a un número en BCD con la finalidad de poder desplegarlo en la pantalla de la LCD. Es por ello que el primer paso de esta rutina es preguntar si el número a comparar no es igual a 0xff lo cual indica que el número es mayor a 2 metros y se tendría que hacer otra acción para ello, De no ser mayor a 200 o igual a 0xff se procede a realizar un rol con la finalidad de poseer un valor proporcional a la distancia recorrida. Ahora para obtener un valor en BCD se procede a realizar restas sucesivas para obtener unidades, decenas y centenas. Una vez adquiridos estos valores se procede a guardar la información en la RAM de display a partir de la dirección 0x65.

Lruc: En esta subrutina se procede a realizar el promedio de dos valores adquiridos en la unidad de captura para obtener valor de la distancia más cercano al valor real. Para esto es importante entender que la unidad de captura es empleada con la finalidad de obtener el ancho de pulso en alto de la señal echo. Para ello se vale de los cambios de estado detectados para que al restarlos nos dan justamente el valor del ancho de pulso en cuestión. Este proceso se realiza dos veces para que posterior a ello se realce una suma, para ello el primer valor adquirido se guarda en una dirección de RAM (0x6D) mientras que la otra se queda en un registro. Posterior a esto se suman los dos valores y se divide entre dos, para este proceso es necesario realizar un rol al valor para adquirir el promedio. Al final se compara con 100, de ser mayor se carga a R16 el valor de 0xFF que indica en otra rutina del programa que el valor es mayor a dos metros y se despliega en la LCD esta indicación.

Cabe mencionar que los valores obtenidos mediante la unidad de captura son valores que están dentro de 8 bits por lo que se puede descartar la parte alta del mismo.

Buzzers: En esta subrutina se carga el valor del OCR1AH y OCR1AL para generar la frecuencia deseada del buzzer dependiendo de la distancia, es decir si es menor a 1 metro se generará una frecuencia de 500[Hz], si es menor a 50 cm generará una frecuencia de 10[Hz] y si es menor a 30 cm se generará una frecuencia de 5 [Hz]. Dichos valores fueron calculados previamente y almacenados en RAM.

V. CONCLUSIONES

- El proyecto se concluyó de forma exitosa al haberse terminado en el tiempo establecido y al haber entregado como resultado un sistema electrónico funcionando con un 95% de efectividad.
- Algunos detalles a mejorar en la programación de este proyecto son el control y discriminación de las señales de eco que recibe el sensor ultrasónico y que son leídas por la unidad de captura del microcontrolador, la generación de las señales con PWM para el control del servomotor y el control de movimiento del sensor ultrasónico por medio del joystick y el convertidor analógico digital.
- Respecto al hardware, valdría la pena explorar el funcionamiento de otros sensores de distancia más precisos y sencillos de controlar, además de implementar sensores de otro tipo como acelerómetros o sensores de color que permitan generar una aplicación más seria de este proyecto, como por ejemplo la generación de un sistema primitivo de un ojo biónico.
- Aunque el sistema desarrollado se ve sencillo, la dificultad para su implementación es alta al haberse utilizado prácticamente todos los bloques funcionales del microcontrolador ATMEGA16 como lo fueron la unidad USART, el ADC, los tres temporizadores, la unidad de captura, las interrupciones externas y por temporizadores, y al haber creado un sistema multifuncional con la ejecución coordinada de distintas operaciones en paralelo.

VI. PROGRAMA EN LENGUAJE ENSAMBLADOR

```
//Funcionamiento de servomotor
.include "m16def.inc"

//Declaración de variables
.equ oc2 = 7
.equ ptccr2 = 0b01101111          //Palabra de control de Timer 2; Pre-escala de 1024, Fast PWM, modo no invertido
.equ ptccr0 = 0b00001101
.equ pgicr = 0b10000000           //Palabra de control para interrupciones externas. Activar INT1
.equ pmcucr= 0b00001100 //Variable para definir palabra de control para interrupciones INT0 e INT1. INT1 con flanco de subida
.equ ptimsk = 0b01110010 //Palabra de control para interrupciones por timer. Activar TMR1 UCP, TMR2 Compare, TMR0 Compare
.equ itcnt2 = 230
.equ neutral = 241                //Para generar duty cycle de x%, posición neutral de servomotor
.equ minimo = 234                 //Para generar duty cycle de y%, posición inicial
.equ maximo = 248                 //Para generar duty cycle de z%, posición final
.equ ptccr1a = 0b00000000         //Palabra para registro TCCR1A, modo normal
.equ ptccr1b = 0b01000101   //Palabra para registro TCCR1B, pre-escala 1024, modo normal, flanco de subida unidad de captura
.equ ptccr1a2 = 0b00000000        //Palabra para registro TCCR1A, modo CTC, salidas deshabilitadas
.equ ptccr1b2 = 0b00001101        //Palabra para registro TCCR1A, modo CTC, pre-escala de 1024
.equ pocr1al5Hz = low(781)
.equ pocr1ah5Hz = high(781)       //Palabra para generar frecuencia de PWM de 5 Hz
.equ pocr1al10Hz = low(390)
.equ pocr1ah10Hz = high(390)      //Palabra para generar frecuencia de PWM de 10 Hz
.equ pocr1al500Hz = low(8)
.equ pocr1ah500Hz = high(8)       //Palabra para generar frecuencia de PWM de 500 Hz

.equ pucsrc = 0b10001110          //Palabra para registro UCSRC
.equ pucsrb = 0b00001000          //Palabra para registro UCSRB
.equ pubrrl = 34                  //Palabra para registro UBRRL
.equ padmux = 0b11000100          //Palabra para registro ADMUX
.equ padcsra = 0b10000111         //Palabra para registro ADCSRA
.equ changeedge = 0b01000000      //Palabra para registro TCCR1B que permite modificar sensibilidad de unidad de captura
.equ tecla = 0x69                 //Registro que almacena código de tecla presionada
.def estadoBF =   r0              //Registro que sirve como almacén de nibble que contiene señal Busy de LCD
.def psreg = r1                   //Registro que sirve como almacén temporal de registro sreg
.def xortccr1b =  r2              //Registro que sirve como código para toggle de sensibilidad de tccr1b
.def picr1 = r3                   //Registro que sirve como almacen de contador de unidad de captura
.def pulsossensor = r4            //Registro que sirve como control de pulsos a detectar
.def motorfijo =  r7              //Registro que sirve como almacenador de código de validación de movimiento de servo
.def palabra1 =   r8              //Registro que sirve como operador de pulso más lejano
.def palabra2 =   r9              //Registro que sirve como operador de pulso más próximo
.def actbuzzer =  r10             //Registro que sirve como modificador de estado de pines de puerto A
.def general =    r16             //Registro que sirve como registro general
.def general1 =   r17             //Registro que sirve como registro de contador
.def cont1seg =   r18             //Registro que sirve como contador de TMR0
.def posservo =   r19             //Registro que sirve como controlador de posiciones de servomotor
.def estados =    r20             //Registro que sirve como controlador de opciones de teclado
.def datosLCD =   r21             //Registro que sirve como controlador de datos/instrucciones enviadas a LCD
.def contador =   r22             //Registro que sirve como contador de eventos generales en el programa
.def contador1 =  r23             //Registro que sirve como contador auxiliar de eventos en el programa
.def contador2 =  r24             //Registro que sirve como contador para tranmisión de datos en comunicación serial
.def actbuzzer1 = r25             //Registro que sirve como transporte del modificador de estado de pines de puerto A

.equ RS = 0                       //Variable para definir pin donde está señal RS de LCD
.equ RW = 1                       //Variable para definir donde está señal RW de LCD
.equ En = 2                       //Variable para definir donde está señal E de LCD
.equ BF = 7                       //Variable para definir donde está señal Busy de LCD
```

```
.equ LCD = PORTB                    //Variable para definir puerto de control de LCD

//Definición instrucciones de LCD
.equ iniLCD4bits = 0x20             //Instrucción para LCD en modo de 8 bits que configura modo de trabajo en 4 bits
.equ clearscreen = 0x01             //Instrucción para LCD que limpia pantalla
.equ returnhome = 0x02              //Instrucción para LCD que reinicia pantalla
.equ functionset = 0b00101000       //Instrucción para LCD que configura modo de trabajo de pantalla
                                    //DL(4) = 1-8b/0-4b; N(3)=1-2L/0-1L; F(2)=1-5x10/0-5x8
.equ entrymodeset = 0b00000110      //Instrucción para LCD que configura cursor y desplazamiento de pantalla
                                    //I/D(1)=1-(->)/0-(<-); SH(0)=1-pantalla móvil
.equ displaycontrol = 0b00001100    //Instrucción para LCD que enciende pantalla, cursor y parpadeo.
                                    //D(2)=Pantalla; C(1)=Cursor; B(0)=Parpadeo de cursor
.equ primeralinea = 0b10000000      //Instrucción para colocar escritura de LCD en inicio de primera línea
.equ segundalinea = 0b11000000      //Instrucción para colocar escritura de LCD en inicio de segunda línea
.equ movercursor = 0b00010100       //Instrucción para desplazar una posición a la derecha al cursor de LCD

//Vectores de interrupción
.org 0x00
jmp main                            //Interrupción de reset
.org 0x04
jmp teclado                         //Interrupción de teclado EXT1
.org 0x08
jmp frecuencia                      //Interrupción de frecuencia de 200 Hz con TMR2 por overflow
.org 0x0A
jmp tmr1c                           //Interrupción de unidad de captura
.org 0x0C
jmp tmr1ca                          //Interrupción de compare A
.org 0x26
jmp cambio                          //Interrupción de 5 ms para generar retardo de 1 s

//Inicio de programa
.org 0x30
main:
        ldi r16,low(ramend)
        out spl,r16
        ldi r16,high(ramend)
        out sph,r16                 //Inicio de pila

//Definición de puertos
        clr r16
        out portd,r16               //Puerto D de salida inicia en estado bajo
        cbi pind,6                  //PD6 inicia en estado bajo
        cbi pind,3                  //PD3 inicia en estado bajo
        ldi r16,0b10100110
        out ddrd,r16                //PD7,PD5,PD2,PD1 como salidas

        clr r16
        out porta,r16               //Puerto A inicia en estado bajo
        ldi r16,0b11100000
        out ddra,r16                //PA5-PA7 como salidas
        clr r16
        out portb,r16               //Estado inicial de puerto B en estado bajo
        ser r16                                 //
        out ddrb,r16                //Puerto B como salida
```

```
//Programa principal
//LCD
	call LCDinit8bitsTo4bits  //Configurar LCD en modo de 8 bits a modo de 4 bits
	call initLCD4bits		//Configurar estado inicial de LCD en 4 bits

//Registros de interrupciones
	ldi general,pgicr
	out gicr,general		//Activar interrupción INT1 (teclado)
	ldi general,pmcucr
	out mcucr,general		//Configurar INT1 con flanco de subida
	ldi r16,ptimsk
	out timsk,r16		//Habilitar interrupciones por timer: Overflow TM2 (servomotor), Compare TMR0 (activar sensor), Unidad de captura TMR1 (leer distancia)

//Registros de comunicación serial
	ldi r16,pucsrc
	out ucsrc,r16			//Habilitar comunicación serial, sin paridad, 1 bit de parada, 1 byte para palabra
	ldi r16,pubrrl
	out ubrrl,r16			//Configurar baud rate de 14 400 baudios/s

//Registro de ADC
	ldi r16,padmux
	out admux,r16			//ADC con palabras no invertidas, voltaje de referencia interno de 2.56 V, entrada de A4
	ldi r16,padcsra
	out adcsra,r16			//ADC habilitado, sin interrupción, pre-escala de conversión de 128

//Registros de control de operación
	ldi r16,itcnt2
	out tcnt2,r16			//Iniciar contador de timer2 en valor inicial para generar 300 Hz de frecuencia
	ldi r16,minimo
	sts 0x60,r16			//Almacenar en 0x60 valor de posición mínima de servomotor
	ldi r16,neutral
	sts 0x61,r16			//Almacenar en 0x61 valor de posición neutral de servomotor
	ldi r16,maximo
	sts 0x62,r16			//Almacenar en 0x62 valor de posición máxima de servomotor
	ldi r16,minimo
	sts 0x64,r16			//Almacenar en 0x64 valor de posición actual de servo motor
	out ocr2,r16			//Cargar a OCR2 dato de posición inicial de servo motor (mínima)
	ldi r16,77
	out ocr0,r16			//Cargar a OCR0 dato para retardo de 10 ms (valor fijo)
	ldi r16,' '
	sts 0x65,r16			//Almacenar caracter inicial en visualización de datos de distancia (' ' si dist<200 cm, '>' si dist>200 cm)
	clr r16
	sts 0x66,r16			//Almacenar valor inicial de 0 para visualización de unidades de cm
	sts 0x67,r16			//Almacenar valor inicial de 0 para visualización de decenas de cm
	sts 0x68,r16			//Almacenar valor inicial de 0 para visualización de centenas de cm
	ldi r16,neutral
	sts 0x89,r16			//Establecer que posición de servo motor en modo prueba sea la neutral
	ldi r16,3
	sts tecla,r16			//Almacenar valor inicial de 3 para registro que almacena valor de tecla presionada
	ldi r18,200				//Generar contador para dos segundos con TMR0
	ldi r19,3			//Generar contador para controlar posiciones de servo motor (3 giros)
	ldi r16,changeedge
	mov r2,r16			//Generar registro de cambio de sensibilidad para unidad de captura (cambiar el tipo de flanco a detectar)
```

```
        ldi r16,3
        mov r7,r16          //Crear controlador para que motor se detenga cuando haya dado tres giros
        clr r16
        mov r5,r16          //Iniciar contador de capturas hechas en modo prueba
        mov r10,r16         //Iniciar registro que contiene control de cambio de estado en pin de buzzer
        ldi xl,low(0x61)
        ldi xh,high(0x61)   //Puntero para control de posición de servo motor, inicia en dirección de posición neutra
        ldi yl,low(0x100)
        ldi yh,high(0x100)  //Puntero para control de captura de TMR1; los datos se colectan a partir de la dirección
0x100 de SRAM

        sei                 //Activar interrupciones globales

//Sistema apagado
fin:
        lds r16,tecla
        cpi r16,0x0D            //Tecla '0' para encender sistema
        brne fin                //Mientras no se presione tecla '0' el sistema está apagado
        ldi r16,0xFF
        sts tecla,r16
        call mensajeinicial     //Cuando se encienda el sistema, mostrar mensaje de inicio

//Sistema encendido
selfuncion:
        lds r16,tecla           //Preparar código de tecla presionada para comparar
        cpi r16,0x0C
        breq apagar                     //Tecla '*' para apagar sistema
        cpi r16,0x00
        breq debilvisual1   //Tecla '1' para modo ceguera
        cpi r16,0x01
        breq debilvisual1   //Tecla '2' para modo débil visual 1
        cpi r16,0x02
        breq debilvisual1   //Tecla '3' para modo débil visual 2
        cpi r16,0x04
        breq prueba2            //Tecla '4' para modo prueba
        cpi r16,0x07
        breq mensajes2          //Tecla 'B' para mostrar mensajes en LCD
        rjmp selfuncion         //Si al principio no se presionó ninguna tecla, no hacer nada hasta que se seleccione
modo

prueba2:
        jmp prueba
mensajes2:
        jmp mensajes

debilvisual1:
        jmp debilvisual2

/*****************************************************************************************************
********/
apagar:
        ldi r16,0x10
        sts tecla,r16                           //Reiniciar registro de tecla
        clt
        ldi datosLCD,clearscreen
        call WriteDataInstruction4Bits  //Borrar LCD
```

```
        clr r16
        out tccr2,r16
        out tccr1b,r16
        out tccr0,r16                                   //Apagar temporizadores

        ldi r16,minimo
        sts 0x64,r16                                    //Reiniciar posición de servomotor
        out ocr2,r16
        ldi r16,itcnt2
        out tcnt2,r16                                   //Reiniciar contador de TMR2
        ldi r19,3                                       //Reiniciar contador de giros de servo motor
        clr r16
        out tcnt0,r16                                   //Reiniciar contador de TMR0
        out tcnt1h,r16
        out tcnt1l,r16                                  //Reiniciar contador de TMR1


        jmp fin                                                 //Vuelve para esperar que se encienda el sistema de nueva
cuenta
/*****************************************************************************************************************
******/
prueba:
        lds r16,tecla                                   //Preparar código de tecla para comparar
selfuncion2:
        cpi r16,0x00
        breq dbv11
        cpi r16,0x01
        breq dbv11
        cpi r16,0x02
        breq dbv11
        cpi r16,0x04
        breq pruebanew
        cpi r16,0x07
        breq mensajes11
        cpi r16,0x0C
        breq apagar11
        lds r16,0x70
        rjmp selfuncion2

dbv11:
        jmp dbv1
mensajes11:
        jmp mensajes
apagar11:
        jmp apagar

pruebanew:
        sts 0x70,r16                    //Almacenar código de tecla presionada
        clr r12                                 //Reiniciar contador de capturas realizadas
        cbi ddra,7

        clt
        ldi datosLCD,clearscreen
        call WriteDataInstruction4Bits  //Borrar pantalla LCD

        ldi r16,ptccr1a
        out tccr1a,r16
```

```
        clr r16
        out tccr1b,r16
        out tccr0,r16                                   //Apagar temporizadores TMR1,TMR0
        out tcnt0,r16
        out tcnt1h,r16
        out tcnt1l,r16                                  //Reiniciar contadores de temporizadores

//Conversión AD para joystic
        sbi adcsra,adsc                 //Iniciar conversión
esperar:
        sbis adcsra,adif            //Esperar a que haya finalizado conversión
        rjmp esperar
        sbi adcsra,adsc                 //Cuando haya finalizado conversión, deshabilitar conversión
                        //0x85=lowori 0x86=highori 0x87=lownew 0x88=highnew

//Lectura de ADC
        in r17,adcl                             //Leer palabra baja de dato convertido
        in r16,adch                             //Leer palabra alta de dato convertido
        cpi r16,0b00000011              //Comparar con 0x03
        breq mayorigual                 //Si es igual a 0x03, código es igual o mayor a 900, verificar que dato es
        clr r17
        ldi r16,minimo                  //Si no es igual a 0x03, código es menor a 900,
        sts 0x89,r16                    //entonces preparar servo motor para que esté en posición mínima
        rjmp sigueprueba

mayorigual:
        cpi r17,0b10110110              //Comparar parte baja de conversión con byte menos significativo de 900
        brlo posneu                             //Si es menor a 900, modificar posicion de servo motor
        clr r17
        ldi r16,maximo
        sts 0x89,r16
        rjmp sigueprueba*/

posneu:
        clr r17
        ldi r16,neutral
        sts 0x89,r16

sigueprueba:
        lds r16,0x89
        sts 0x64,r16                                    //Fijar posición de servomotor
        ldi yl,low(0x100)
        ldi yh,high(0x100)                              //Preparar puntero para almacenar información

        ldi r16,ptccr2
        out tccr2,r16                   //Iniciar TMR2, fijación de servomotor en una dirección
        ldi r16,ptccr1b
        out tccr1b,r16                  //Iniciar TMR1, unidad de captura
        ldi r16,ptccr0
        out tccr0,r16                   //Iniciar TMR0, cada segundo activará sensor ultrasónico

ldpfsm:
        mov r25,r12                             //Preparar número de capturas realizadas para comparación
        cpi r25,16                              //Comparar con número 16
        breq edpfsm                             //Si es menor a 16, esperar a que se hayan hecho 16 capturas
        rjmp ldpfsm
```

```
edpfsm:
        clr r16
        out tccr0,r16                   //Apagar TMR0, detener activación de sensor
        out tccr2,r16
        out tccr1b,r16                  //Apagar TMR1, detener unidad de captura
        out tcnt1h,r16
        out tcnt1l,r16                  //Reiniciar valor de contador de timer1

//Desplegar información en LCD
        ldi r24,32
        ldi zl,low(0x400<<1)                    //Preparar puntero z para lectura en ROM de plantilla de mensaje de distancia
        ldi zh,high(0x400<<1)
        clt
        ldi datosLCD,primeralinea       //Colocar puntero de LCD en primera posición de primera línea
        call WriteDataInstruction4Bits

        ldi contador,16
escribir:
        lpm datosLCD,z+
        set
        call WriteDataInstruction4Bits          //Escribir sobre toda la línea 1 de LCD
        dec contador
        brne escribir

        clt
        ldi datosLCD,0x8A
        call WriteDataInstruction4Bits          //Colocar puntero en posición adecuada para escribir distancia medida por sensor

        ldi yl,low(0x118)
        ldi yh,high(0x118)                                      //Preparar apuntador y para obtener promedio

        call lruc                                               //Rutina para obtener promedio de 8 señales leídas
        clr r8
        clr r9
        call prepararnumeros                            //Rutina de conversión de binario a BCD
        ldi zl,low(0x500<<1)
        ldi zh,high(0x500<<1)           //Preparar puntero z para mensaje de dirección
        call escribirLCD                //Escribir distancia detectada por sensor; en caso de que se encuentre en modo débil visual 2, mostrar dirección en segunda línea

        call retardo1s


//Enviar información por comunicación serial

        ldi yl,low(0x100)
        ldi yh,high(0x100)              //Colocar puntero y en dirección de SRAM para leer valores capturados
        ldi r24,32                              //Preparar contador de cantidad de valores a enviar por puerto serial

        ldi r16,pucsrb
        out ucsrb,r16                   //Habilitar transmisión serial de microcontrolador

trans_ser:
        ld r16,y+
        out udr,r16                             //Enviar por puerto serial primer dato capturado
```

```
wait2:
        sbis ucsra,txc
        rjmp wait2
        sbi ucsra,txc
        dec r24
        breq cont_prueba
        rjmp trans_ser              //Si no es igual a cero, esperar

cont_prueba:
        clr r16
        out ucsrb,r16               //Apagar comunicación serial
        jmp prueba

/*****************************************************************************************************************
*****/
mensajes:

        clr r16
        out tccr2,r16
        out tccr1b,r16
        out tccr0,r16                                   //Apagar temporizadores

        ldi r16,minimo
        sts 0x64,r16                                    //Reiniciar posición de servomotor
        out ocr2,r16
        ldi r16,itcnt2
        out tcnt2,r16
        ldi r19,3
        clr r16
        out tcnt0,r16
        out tcnt1h,r16
        out tcnt1l,r16

        ldi r16,0x10
        sts tecla,r16                                   //Reiniciar registro de tecla

        clt
        ldi datosLCD,clearscreen
        call WriteDataInstruction4Bits      //Borrar LCD

        call mensajeinicial

        jmp selfuncion
/*****************************************************************************************************************
*******/
debilvisual2:
        lds r16,tecla
selfuncion1:
        cpi r16,0x00
        breq dbv1
        cpi r16,0x01
        breq dbv1
        cpi r16,0x02
        breq dbv1
        cpi r16,0x04
        breq prueba1
        cpi r16,0x07
```

```
        breq mensajes1
        cpi r16,0x0C
        breq apagar1
        lds r16,0x70
        rjmp selfuncion1

apagar1:
        jmp apagar
mensajes1:
        jmp mensajes
prueba1:
        jmp prueba

dbv1:
        clr r12
        sts 0x70,r16
        sbi ddra,7

        clt
        ldi datosLCD,clearscreen
        call WriteDataInstruction4Bits

        ldi r16,ptccr1a
        out tccr1a,r16
        ldi r16,ptccr1b
        out tccr1b,r16                  //Iniciar TMR1

        ldi r16,ptccr0
        out tccr0,r16                   //Iniciar TMR0
        ldi r16,ptccr2
        out tccr2,r16                   //Iniciar TMR2

esperar12:
        cp r4,r7                        //Verificar que se hayan cargado código de 3 posiciones
        cbi porta,7
        brne esperar12                  //Si no está el código, sigue esperando

        clr r16                                 //
        mov r4,r16
        out tccr0,r16                   //Apagar TMR0, control de cambio de posición de motor y activación de sensor
        out tccr1b,r16                  //Apagar TMR1, unidad de captura
        out tccr2,r16                   //Apagar TMR2, generación de señal PWM
        out tcnt1h,r16
        out tcnt1l,r16                  //Reiniciar valor de contador de timer 1

        //Pulsos se almacenan desde 0x100
        ldi yh,high(0x100)
        ldi yl,low(0x100)
        call lruc

        sts 0x6A,r16                    //Distancia a la derecha
        ldi yh,high(0x108)
        ldi yl,low(0x108)
        call lruc

        sts 0x6B,r16                    //Distancia al frente
        ldi yh,high(0x110)
```

```
        ldi yl,low(0x110)
        call lruc

        sts 0x6C,r16                    //Distancia a la izquierda

        //Mensaje: Dirección, X metros
        ldi r16,ptccr1a2
        out tccr1a,r16                  //Preparar TMR1, modo CTC con pre-escala de 1024

        lds r16,0x70                    //Preparar código de tecla para comparación
        cpi r16,0x00                    //Comprobar que código pertenece a modo ceguera
        breq ceguera0                   //Si lo es, no actives LCD y pasa a buzzers

        clr r8                  //Si es cualquier modo de débil visual, preparar registros para escritura de datos en LCD
        clr r9
        ldi zl,low(0x400<<1)    //Preparar puntero z para lectura en ROM de plantilla de mensaje de distancia
        ldi zh,high(0x400<<1)
        clt
        ldi datosLCD,primeralinea       //Colocar puntero de LCD en primera posición de primera línea
        call WriteDataInstruction4Bits

        ldi contador,16
escribir6:
        lpm datosLCD,z+
        set
        call WriteDataInstruction4Bits          //Escribir sobre toda la línea 1 de LCD
        dec contador
        brne escribir6

        clt
        ldi datosLCD,0x8A
        call WriteDataInstruction4Bits  //Colocar puntero en posición adecuada para escribir distancia medida por sensor

        lds r16,0x6A                            //Leer valor en binario adquirido por sensor en dirección derecha
        call prepararnumeros                    //Rutina de conversión de binario a BCD
        ldi zl,low(0x500<<1)
        ldi zh,high(0x500<<1)                   //Preparar puntero z para mensaje de dirección
        subi zl,-32
        call escribirLCD        //Escribir distancia detectada por sensor; en caso de que se encuentre en modo débil visual 2,
mostrar dirección en segunda línea

        lds r16,0x70
        cpi r16,0x02                                    //Verificar que no se encuentra en modo débil visual 2
        breq sinbuzzer0                                 //Si es modo débil visual 2, no activar buzzer

ceguera0:
        lds r16,0x6A                            //Leer dato en bruto medido por sensor
        ldi r25,0b00100000                      //Preparar código para modificar estado de pin de buzzer 1
        mov r10,r25                             //Guardarlo en registro adecuado
        call buzzers                            //Rutina de encendido de buzzer y generación de PWM para sonido

sinbuzzer0:
        call retardo1s                                  //Generar pausa de 1 segundo
        clr r25
        clr r16
        out tccr1b,r16                                  //Apagar TMR1
        out tcnt1h,r16
```

```
        out tcnt1l,r16                                  //Reiniciar contador de TMR1
        cbi porta,5                                             //Colocar en estado bajo pin5 de puerto A

        lds r16,0x70                                    //Preparar código de modo en funcionamiento
        cpi r16,0x00                                    //Verificar si código pertecene a modo ceguera
        breq ceguera1                                   //Si se encuentra en modo ceguera, saltar escritura en LCD

        clt                     //Si no se encuentra en modo ceguera, colocar puntero para escribir dígitos de distancia
        ldi datosLCD,0x8A
        call WriteDataInstruction4Bits

        lds r16,0x6B                                    //Valor de distancia en dirección frontal
        call prepararnumeros                            //Rutina de conversión de binario a BCD
        ldi zl,low(0x500<<1)
        ldi zh,high(0x500<<1)                           //Preparar puntero para escribir dirección frontal
        subi zl,-48
        call escribirLCD                        //Escribir número en la LCD, además de la dirección si fuera necesario

        lds r16,0x70                            //Preparar código de modo en funcionamiento
        cpi r16,0x02                            //Verificar si código pertecene a modo débil visual 2
        breq sinbuzzer1                         //Si se encuentra en modo débil visual 2, saltar activación de buzzers

ceguera1:
        lds r16,0x6B                            //Preparar datos en bruto de distancia en dirección frontal
        ldi r25,0b01000000                      //Preparar código de activación de pin A6
        mov r10,r25                             //Preparar registro para código de activación de PA6
        call buzzers                            //Rutina de activación de buzzers

sinbuzzer1:
        call retardo1s                          //Generar pausa de 1 segundo
        clr r25
        clr r16
        out tccr1b,r16                          //Apagar TMR1
        out tcnt1h,r16                                  //
        out tcnt1l,r16                          //Reiniciar contador de TMR1
        cbi porta,6                             //Colocar en estado bajo PA6

        lds r16,0x70
        cpi r16,0x00
        breq ceguera2

        clt
        ldi datosLCD,0x8A
        call WriteDataInstruction4Bits

        lds r16,0x6C
        call prepararnumeros
        ldi zl,low(0x500<<1)
        ldi zh,high(0x500<<1)
        subi zl,-64
        call escribirLCD

        lds r16,0x70
        cpi r16,0x02
        breq sinbuzzer2

ceguera2:
```

```
        lds r16,0x6C
        ldi r25,0b10000000
        mov r10,r25
        call buzzers
sinbuzzer2:
        call retardo1s
        clr r25
        clr r16
        out tccr1b,r16
        out tcnt1h,r16
        out tcnt1l,r16
        cbi porta,7
        nop
        nop

        ldi xl,low(0x61)
        ldi xh,high(0x61)
        ldi r16,minimo
        sts 0x64,r16
        ldi yl,low(0x100)
        ldi yh,high(0x100)
        ldi r19,3
        ldi r16,itcnt2
        out tcnt2,r16
        clr r16
        out tcnt0,r16
        cbi porta,5
        cbi porta,6
        cbi porta,7

        jmp debilvisual2

//Interrupciones
/*************************************************************************************************************
********/
tmr1ca:
        in r1,sreg
        push r1
        push r16

        in r16,porta
        eor r16,r10
        out porta,r16

        pop r16
        pop r1
        out sreg,r1
        reti

/*************************************************************************************************************
********/
teclado:
        in r1,sreg
        push r1
        push r16

        in r16,pina
```

```
        andi r16,0x0F
        sts tecla,r16                           //Almacenar opción leída de teclado

        pop r16
        pop r1
        out sreg,r1
        reti
/*****************************************************************************************************
*********/
cambio:
        in r1,sreg
        push r1
        push r16
        sei                                     //Habilitar interrupción para no perder PWM
        dec r18                                 //Verificar que no ha pasado 1 segundo
        breq continuar                          //Si ya ha pasado 1 segundo, continua con interrupción
        cpi r18,100                             //Verificar que no ha pasado 1/2 segundo
        brne salir0                             //Si no ha pasado 1/2 segundo, salir de interrupción
        sbi portd,5                             //Si ha pasado 1/2 segundo, activar sensor ultrasónico
        call retardo10us
        cbi portd,5
        rjmp salir0                             //Salir de interrupción
continuar:                                      //Si ha pasado 1 segundo...
        ldi r18,200
        sbi portd,5
        call retardo10us
        cbi portd,5                             //Activar sensor ultrasónico
        call retardo10ms                    //Esperar 10 ms en posición final para recibir último registro de sensor

        lds r16,0x70
        cpi r16,0x04                            //Verificar que sistema no se encuentra en modo prueba
        breq salir0

        ld r16,x+                               //Leer nueva posición para servo motor almacenada en RAM
        sts 0x64,r16                            //Almacenarla en 0x64 para lectura
        dec r19                                 //Verificar que no se han utilizado 3 posiciones
        brne salir0
        ldi r16,3
        mov r4,r16
        ldi r19,3                               //Si ya se ocuparon 4 posiciones, reinicia contador de posiciones
        ldi xl,low(0x60)
        ldi xh,high(0x60)                   //Seleccionar posición inicial de servomotor

salir0:
        pop r16
        pop r1
        out sreg,r1
        reti
/*****************************************************************************************************
**********/
frecuencia:
        in r1,sreg
        push r1
        push r16

        lds r16,0x64
        out ocr2,r16
```

```
        ldi r16,itcnt2
        out tcnt2,r16       //Evitar que contador de TMR2 inicie en cero y comience en 230 para generar frecuencia de 300 Hz

        pop r16
        pop r1
        out sreg,r1
        reti
/*****************************************************************************************************************
**********/
tmr1c:
        in r1,sreg
        push r1
        push r16

        in r3,icr1l                         //Leer valor bajo capturado por unidad de captura
        st y+,r3                            //Almacenar a partir de 0x100 de RAM
        in r3,icr1h                         //Leer valor alto capturado por unidad de captura
        st y+,r3                            //Almacenar a partir de 0x101 de RAM

        in r16,tccr1b
        eor r16,r2                          //Modificar sensibilidad de unidad de captura
        out tccr1b,r16
        ldi r16,(1<<icf1)
        out tifr,r16                        //Apagar bandera de interrupción por unidad de captura

        inc r12

salir1:
        pop r16
        pop r1
        out sreg,r1
        reti
/*****************************************************************************************************************
*************/

//Subrutinas
/************Condición de Busy Flag en 8 bits*********************/
WaitBusyFlag8bits:
        cbi LCD,En
        cbi LCD,RS
        sbi LCD,RW          //Iniciar LCD en modo lectura de instrucción, RS=0,RW=1,E=0
leer1:
        sbi LCD,En          //Iniciar lectura de LCD
        nop
        nop
        nop
        in estadoBF,pinb    //Almacenar lectura de LCD en registro
        cbi LCD,En          //Terminar lectura de LCD
        sbrc estadoBF,BF    //Analizar si Busy Flag está en estado bajo
        jmp leer1           //Si no lo está, vuelve a leer estado de LCD
        cbi LCD,RW          //Si lo está, termina modo lectura de instrucción, RS=0,RW=0,E=0
        ret
/************Condición de Busy Flag en 4 bits*********************/
WaitBusyFlag4bits:
        cbi LCD,En
        cbi LCD,RS
        sbi LCD,RW                  //Iniciar LCD en modo lectura de instrucción, RS=0,RW=1,E=0
```

```
leer2:
        sbi LCD,En                      //Iniciar lectura de primer nibble de LCD
        nop
        nop
        nop
        in estadoBF,pinb                //Almacenar primer nibble de LCD en registro
        push estadoBF                   //Guardar nibble en pila
        cbi LCD,En                      //Terminar lectura de primer nibble
        sbi LCD,En                      //Iniciar lectura de segundo nibble de LCD
        nop
        nop
        nop
        in estadoBF,pinb                //Almacenar segundo nibble de LCD en registro
        cbi LCD,En                      //Terminar lectura de segundo nibble
        pop estadoBF                    //Regresar desde pila a primer nibble almacenado
        sbrc estadoBF,BF        //Checar si Busy Flag está en estado bajo
        jmp leer2                       //Si no lo está, regresa a leer estado de LCD

        cbi LCD,RW                      //Si lo está, stermina modo lectura de instrucción,RS=0,RW=0,E=0
        ret
/***********Inicializar LCD a 4 bits desde configuración de 8 bits********/
LCDinit8bitsTo4bits:
        push general

        call WaitBusyFlag8bits          //Leer estado de Busy Flag
        ldi datosLCD,iniLCD4bits//Si LCD está disponible, cargar instrucción de 8 bits para configurar LCD en 4 bits (0x20)

        in general,ddrb                 //
        ori general,0xF0                //
        out ddrb,general                //Configurar como salidas a líneas de datos de LCD
        in general,LCD                          //
        andi general,0x0F               //Colocar en estado bajo pines de datos de LCD sin afectar bits de señales de control
        andi datosLCD,0xF0              //Configurar RS=0,RW=0,E=0 sin afectar nibble alto de dato a ingresar
        or general,datosLCD             //Unir en registro estado de señales de control y estado de línea de datos
        out LCD,general                 //Enviar a puerto de LCD señales de control y dato indicado configuradas
        sbi LCD,En                      //Iniciar escritura de instrucción en LCD
        nop
        nop
        nop
        cbi LCD,En                                      //Terminar escritura de instrucción en LCD

        in general,ddrb                         //
        andi general,0x0F                       //
        out ddrb,general                //Configurar como entradas a líneas de datos de LCD

        pop general
        ret
/***********Escribir dato o instrucción de acuerdo a bandera T***********/
WriteDataInstruction4bits:
        push general                            //Almacenar en pila datos de registro general

        call WaitBusyFlag4bits          //Verificar que LCD está disponible
        cbi LCD,En
        cbi LCD,RS
        cbi LCD,RW                              //Si lo está, configura LCD en modo escritura RS=0,RW=0,E=0
        push datosLCD                   //Guardar instrucción/dato a escribir en pila
        in general,ddrb         //
```

```
        ori general,0xF0        //
        out ddrb,general        //Colocar como salida a 4 líneas de datos de LCD
        in general,LCD          //
        clr general             //Configurar LCD: RS=0,RW=0,E=0,D7-D4=0
        brtc instruccion                //Verificar si bandera T es cero
        sbr general,0b00000001  //Si lo es, coloca en estado alto señal RS, configurar LCD como escritura de datos
                                //Si no lo es, configura LCD como escritura de instrucción
instruccion:
        andi datosLCD,0xF0              //Conservar el nibble alto de instrucción/dato
        or general,datosLCD     //Unir estado de señales de control y de líneas de datos en registro
        out LCD,general         //Enviar a puerto de LCD estado de registro
        sbi LCD,En                      //Iniciar escritura de nibble alto de dato/instrucción
        nop
        nop
        nop
        cbi LCD,En                      //Terminar escritura de nibble alto
        pop datosLCD                    //Recuperar instrucción guardada en pila
        andi general,0x0F       //Borrar nibble alto de dato/instrucción anterior
        swap datosLCD                   //Colocar nibble bajo de instrucción en nibble alto
        andi datosLCD,0xF0              //Conservar el nibble bajo de instrucción/dato
        or general,datosLCD     //Unir estado de señales de control y de líneas de datos en registro
        out LCD,general                 //Enviar a puerto de LCD estado de registro
        sbi LCD,En                      //Iniciar escritura de nibble bajo de dato/instrucción
        nop
        nop
        nop
        cbi LCD,En                      //Terminar escritura de nibble bajo
        in general,ddrb
        andi general,0x0F       //
        out ddrb,general        //Colocar como entradas a líneas de datos de LCD

        pop general                     //Regresa de pila datos originales de registro general
        ret

/***********Inicializar LCD en 4 bits*************************************/
initLCD4bits:
        push general

        clt
        call WaitBusyFlag4bits
        ldi datosLCD,functionset        //Configurar LCD en 4 bits, 2 líneas y caracteres 5x8 puntos
        call WriteDataInstruction4bits
        ldi datosLCD,displaycontrol             //Encender sólo pantalla y cursor
        call WriteDataInstruction4bits
        ldi datosLCD,returnhome                 //Colocar escritura en LCD en primera posición de primera línea
        call WriteDataInstruction4bits
        ldi datosLCD,entrymodeset               //Configurar modo incremental de cursor y desactivar desplazamiento de
pantalla
        call WriteDataInstruction4bits
        ldi datosLCD,clearscreen        //Limpiar pantalla de LCD
        call WriteDataInstruction4bits

        pop general
        ret
/*****************************************************************************/
retardo10us:                                            //Retardo de 10 micro segundos
        push r16
```

```
        push r17

        ldi r16,1
ret2:
        ldi r17,25
ret1:
        dec r17
        brne ret1
        dec r16
        brne ret2

        pop r17
        pop r16
        ret
/*************************************************************************************************/
retardo10ms:                                            //Retardo de 10 micro segundos
        push r16
        push r17

        ldi r16,255
ret7:
        ldi r17,104
ret6:
        dec r17
        brne ret6
        dec r16
        brne ret7

        pop r17
        pop r16
        ret

/*************************************************************************************************/
retardo1s:
        push r16
        push r17
        push r18

        ldi r16,100
ret5:
        ldi r17,255
ret4:
        ldi r18,104
ret3:
        dec r18
        brne ret3
        dec r17
        brne ret4
        dec r16
        brne ret5

        pop r18
        pop r17
        pop r16

        ret
/*************************************************************************************************/
```

```
prepararnumeros:
        cpi r16,0xFF
        brne div0
        ldi r17,'<'
        sts 0x65,r17
        ldi r17,2
        sts 0x68,r17
        rjmp fin1

div0:
        ldi r17,' '
        sts 0x65,r17

        clc
        rol r16                         //Multiplicar por dos
        cpi r16,10                      //Verificar que unidades sean mayores que 10
        brsh div
        sts 0x66,r16                    //Si unidades son menores a 10, almacenar valor y salir
        clr r17
        sts 0x67,r17
        sts 0x68,r17
        rjmp fin1

div:
        subi r16,10                     //Restar de 10 en 10
        inc r8
        cpi r16,10
        brsh div
        sts 0x67,r8                     //Decenas parciales
        sts 0x66,r16                    //Unidades finales

        lds r16,0x67                    //Verificar que decenas sean mayores que 10
        cpi r16,10
        brsh div1
        clr r17
        sts 0x68,r17
        rjmp fin1

div1:
        subi r16,10
        inc r9
        cpi r16,10
        brsh div1
        sts 0x67,r16                    //Decenas finales
        sts 0x68,r9                     //Centenas finales

fin1:
        clr r8
        clr r9

        ret
/*****************************************************************************************************/
escribirLCD:
        set
        lds datosLCD,0x65
        call WriteDataInstruction4Bits
        lds datosLCD,0x68
```

```
        subi datosLCD,-0x30
        call WriteDataInstruction4Bits
        lds datosLCD,0x67
        subi datosLCD,-0x30
        call WriteDataInstruction4Bits
        lds datosLCD,0x66
        subi datosLCD,-0x30
        call WriteDataInstruction4Bits

        lds r16,0x70
        cpi r16,0x02
        brne msjsindir

        clt
        ldi datosLCD,segundalinea
        call WriteDataInstruction4Bits

        ldi contador,16
escribir5:
        lpm datosLCD,z+
        set
        call WriteDataInstruction4Bits
        dec contador
        brne escribir5

msjsindir:
        clr r16
        sts 0x65,r16
        sts 0x66,r16
        sts 0x67,r16
        sts 0x68,r16

        ret
/*************************************************************************************************************
******/
mensajeinicial:
        ldi zl,low(0x500<<1)
        ldi zh,high(0x500<<1)

        clt
        ldi datosLCD,primeralinea
        call WriteDataInstruction4Bits

        ldi contador,2
escritura1:
        ldi contador1,16
escritura0:
        lpm datosLCD,z+
        set
        call WriteDataInstruction4Bits
        dec contador1
        brne escritura0
        dec contador
        breq finalizar
        clt
        ldi datosLCD,segundalinea
        call WriteDataInstruction4Bits
```

```
        rjmp escritura1
finalizar:
        ret
/*****************************************************************************************************************
*****/
lruc:
        ld r8,y                         //Leer parte baja de número 2
        ldd r9,y+2                      //Leer parte baja de número 1
        sub r9,r8                       //Restar partes bajas de número
        sts 0x6D,r9                                     //

        subi yl,-4
        ld r8,y
        ldd r9,y+2
        sub r9,r8

        lds r8,0x6D
        add r8,r9
        clc
        ror r8
        mov r16,r8
        cpi r16,100
        brlo seguir                             //Si el resultado es menor a 100, continuar programa
        ser r16
seguir:
ret
/*****************************************************************************************************************
*****/
buzzers:
        cpi r16,15                              //Comparar valor de distancia con 15
        brlo senal500                   //Si es menor a 15 (< 30 cm) entonces generar ruido de 500 Hz
        cpi r16,25                              //Comparar valor de distancia con 25
        brlo senal10                    //Si es menor a 25 (< 50 cm) entonces generar ruido de 10 Hz
        cpi r16,50                              //Comparar valor de distancia con 50
        brlo senal5                             //Si es menor a 50 (<100 cm) entonces generar ruido de 5 Hz
        clr r16                                 //Si distancia es mayor a 100 cm entonces...
        out tccr1b,r16                  //apagar TMR1
        ret                                             //Salir de subrutina

senal500:
        ldi r16,pocr1ah500Hz            //
        out ocr1ah,r16                  //
        ldi r16,pocr1al500Hz            //
        out ocr1al,r16                  //Preparar registro OCR1A para cambiar estado de puerto A cada 1000 Hz
        rjmp finbuzzers                 //Salir de subrutina
senal10:
        ldi r16,pocr1ah10Hz
        out ocr1ah,r16
        ldi r16,pocr1al10Hz
        out ocr1al,r16
        rjmp finbuzzers
senal5:
        ldi r16,pocr1ah5Hz
        out ocr1ah,r16
        ldi r16,pocr1al5Hz
        out ocr1al,r16
```

```
finbuzzers:
        ldi r16,ptccr1b2
        out tccr1b,r16                  //TMR1 en modo CTC, pre-escala de 1024
        ret

//Memoria ROM
.org 0x400
mensaje:
.db "Distancia:    cm"

.org 0x500
opciones:
.db "0.ON 1.CE 2.CDV1"
.db "3.CDV2 4.CProbe "
.db "    Derecha     "
.db "     Centro     "
.db "   Izquierda    "
```