Universidad de Buenos Aires

Facultad de Ingeniería

Trabajo Práctico Integrador: Decodificador Morse

ALUMNO

Mundani Vegega, Ezequiel **emundani@fi.uba.ar** **102312**

ASIGNATURA

86.07 - Laboratorio de Microprocesadores

15 de diciembre de 2022

Objetivos

En este trabajo se busca programar un microcontrolador ATMega328P en Assembler y hacer su circuito para poder detectar con un sensor de sonido código Morse y su frecuencia, para luego enviar el mensaje recibido a una computadora.

Introducción

El código Morse es un sistema de representación de caracteres mediante señales, en este caso sonoras. Se utilizaron puntos para representar señales cortas y rayas para señales largas. Dado a que la lectura Morse se hizo de manera digital, se pensó que un punto es un 1, una raya son tres 1, el espacio entre caracteres Morse es un 0, el espacio entre letras son tres 0 y el espacio entre palabras son siete 0.

Al momento de leer la información del detector de sonido, como esta se hizo de manera digital se trató a dicha información como una cadena de bits, pudiendo utilizar los 1 y 0 ya mencionados. Esta información será procesada por el microcontrolador para luego enviar letras, números y signos mediante los puretos RX y TX a una computadora.

Como la duración de cada 1 o 0 no es especificada de antemano sino que se debe averiguar, se optó por tener que enviar un 1 (equivalente a una "E" en Morse) seguido de al menos un 0 antes de comenzar cada transmisión. Utilizando una terminal se podrá visualizar la frase, que fue escuchada en Morse, en los caracteres alfanuméricos conocidos.

Lista de materiales

- 1 resistor de 100 kΩ
- 1 diodo LED rojo
- 1 capacitor de 100 nF
- 1 microcontrolador ATmega328P
- 4 cables macho-macho para protoboard
- 1 protoboard
- 1 fuente de alimentación de 5V
- 1 sensor de sonido digital

Desarrollo

Para la realización de este trabajo se decidió dividir el desarrollo en tres partes principales: detección de duración de un punto y lectura de si se está en high o low, flujo de procesamiento de los bits leídos y por último traducción Morse y envío por USART. Para el desarrollo de cada parte se fue probando el código con el circuito, para poder asegurar que lo que se tenía hasta el momento funcionaba de la manera esperada.

Detección de la duración y lectura de PB0

Dado a que lo primero que hay que hacer al recibir los 1 y 0 del sensor de sonido es averiguar su largo, se decidió conectar la salida digital del sensor al puerto 0 del microcontrolador, dado a que este corresponde también con el *input capture pin*. Se utilizó al timer 1 por ser de 16 bits y permitir obtener duraciones de hasta cuatro segundos aproximadamente (siendo el clock del micro es de 16 MHz). Primero se lo configuró para estar detenido, para funcionar en modo normal, para utilizar un prescaler que divide por 1024, para funcionar en modo captura y que la captura ocurra en flanco ascendente.

Una vez que se capturó el primer flanco ascendente, se vuelve a configurar el timer con el mismo prescaler, en modo normal (se lo inicia), pero se cambia su detección a flanco descendente, permitiendo almacenar directamente la duración del primer 1 recibido en el registro ICR1.

Al haber capturado el segundo flanco, se vuelve a configurar el timer con el mismo prescaler, pero ahora en modo CTC, siendo su TOP el valor almacenado en ICR1. Luego de esta configuración se pone al timer en la mitad de tiempo de TOP, de manera que la interrupción por llegar a este ocurra siempre que se está a la mitad de un bit, pudiendo leer de manera confiable su valor.

Cabe aclarar que para la detección por captura se habilitó su interrupción y que cada vez que se llegue a TOP se llamará a la interrupción que leerá el valor del pin B0.

Procesamiento de los bits leídos

Para procesar los bits leídos, se realizó un algoritmo que decide qué hacer en base a la cantidad de 1 o 0 recibidos. Primero se empieza estando en un loop del cual se sale al recibir el primer 1. Al recibir el siguiente bit se ve si es un 1 o 0 y así sucesivamente.

A medida que se van detectando los puntos y rayas, estos se van guardando en la memoria RAM hasta haber recibido una secuencia de tres ceros, que indica que la secuencia recibida por lo menos está bien definida (de si dicha secuencia corresponde a una letra, número o signo se encarga otra parte del programa). Al terminar de almacenar las rayas y puntos se concatena un byte 0x00, correspondiente con el carácter ASCII '\0', que luego será útil para saber cuándo terminar de leer la RAM.

En caso de haber un error se vuelve empieza a procesar la nueva secuencia de unos y ceros, pudiendo esto generar que se envíe un carácter incorrecto o que no exista. Además el código fue pensado para que en caso de querer agregar cierto comportamiento ante un error se pueda hacer sin problema. En caso de haber recibido siete 0, esto indica que se debe enviar un espacio, por lo que directamente se lo envía a través de la USART.

El comportamiento definido puede apreciarse en el siguiente diagrama:

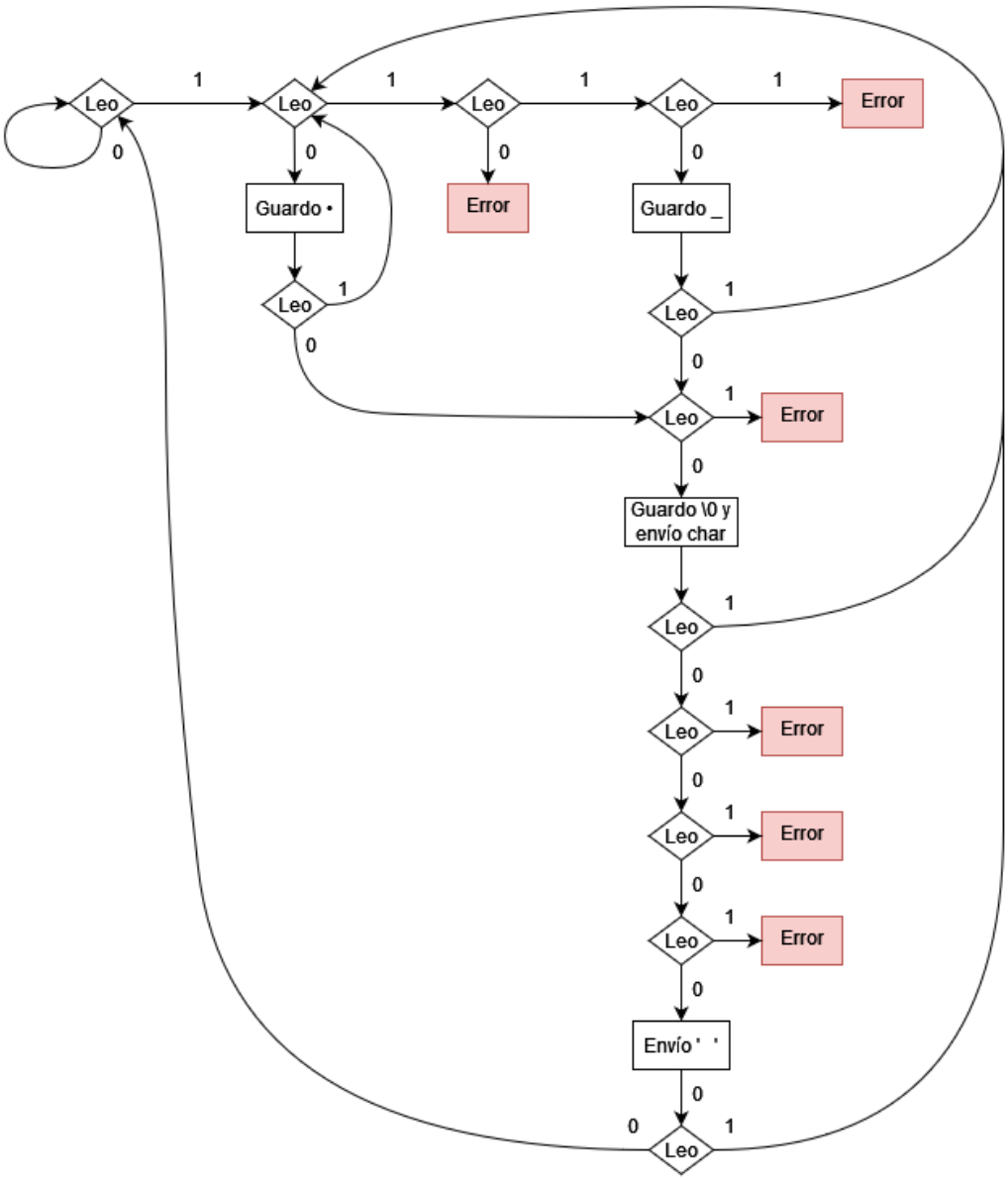

Figura 1: Algoritmo de procesamiento de los bits leidos.

Traducción Morse y envío por USART

Una vez que ya se tiene la cadena de puntos y rayas, se la traduce a la letra, número o signo correspondiente. Para esto se guardó un mapa en la memoria flash, donde cada secuencia correspondiente a cada carácter tiene un '\0' al final. Primero están las letras ordenadas, luego los números del cero al nueve ordenados y finalmente los signos de puntuación.

Cuando se quiere traducir la secuencia de puntos y rayas, se va recorriendo cada secuencia guardada en la flash. En caso de no corresponder algún carácter se pasa a la siguiente secuencia (sabiendo que la siguiente secuencia comienza luego del próximo '\0'. A medida que se va pasando de posible carácter al siguiente, se va incrementando un contador. Una vez que se detectó la secuencia correcta, teniendo el valor contado, y sabiendo que en el mapa hecho y en la tabla ASCII las letras y números están organizadas, si el contador es menor a 26 se trata de una letra, si no es letra y es menor a 36 se trata de un número y si no es un signo de puntuación.

Si es una letra se enviará el carácter ASCII correspondiente al valor del contador más 65 (posición de la "A" en la tabla ASCII). Si es un número se enviará el valor del contador menos las 26 posiciones del mapa de la flash más 48 (posición del "0"). En caso de ser un signo de puntuación, como estos no están ordenados igual que en el mapa hecho, se hace un if-else para ver cuál es.

Una vez detectado lo que se quiere enviar, se guarda en el registro transmitChar. Luego por la USART se enviará lo que haya en él. Para la lectura de lo enviado se utilizó la terminal de Arduino.

Consideraciones especiales

Para la generación del código Morse se utilizó la página web https://rl.se/morse. Antes de su utilización se comprobó que la relación entre las duraciones de los puntos, rayas y espacios sea igual a la pensada para este trabajo. El sonido era reproducido por un parlante y frente a él se conectó el sensor de sonido.

También se conecto la salida digital del sensor al pin analógico A0 y luego se graficó lo leído utilizando la terminal de Arduino. Esto resultó esencial dado a que permitió poder ver cómo afectaban los diferentes componentes al ruido de la señal.

A medida que se hacía el código y se probaba el circuito, se utilizaron LEDs para probar y debugear. En los pines 5, 6 y 7 del puerto D se conectó uno verde, una amarillo y uno rojo respectivamente y se considera que sin los LEDs hubiese sido mucho más difícil la detección de errores y problemas.

Archivos con código

El software fue desarrollado con Microchip Studio, para luego ser ensamblado y subido al microprocesador con AVRDude. El programa consiste en varios archivos ".asm". "main.asm" contiene el flujo básico del programa y el código de las interrupciones. "morseMap.asm" contiene un mapa que estará en la memoria Flash del programa, en el cual se definen todas las combinaciones de puntos y rayas que aceptará el programa, poniendo luego de cada un byte 0x00. "morseSubroutines.asm" poseé toda la lógica para relacionar cada secuencia Morse con su respectiva letra, número o carácter. "morseRead.asm" posee el flujo de qué hacer a medida que se van detectando bits en alto o bajo en la entrada B0. "timersConfiguration.asm" contiene las subrutinas para configurar los timers, como su nombre lo indica. "delays.asm" contiene la lógica de todos los delays utilizados (en este caso uno solo). "USART.asm" tiene las subrutinas de la configuración de la USART y para poder enviar y recibir las transmisiones.

Flujo del programa

El flujo del programa es el siguiente: primero se configuran los puertos, la USART, se inicializan el stack pointer, los punteros para la RAM y los registros a utilizar, luego se configura el timer por primera vez.

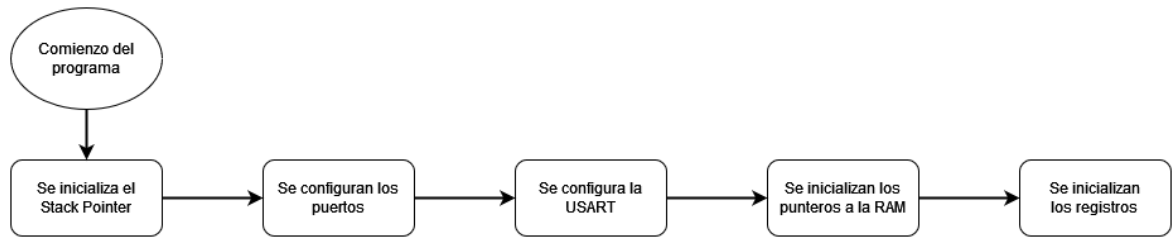

Figura 2: Diagrama del flujo comienzo del programa.

Después hay un bucle en el que no pasa nada y se está a la espera de la interrupción de captura por flanco ascendente. Luego se vuelve a configurar el timer y se está en otro bucle en el que no ocurre nada y se sale al ocurrir la interrupción de captura por flanco descendente.

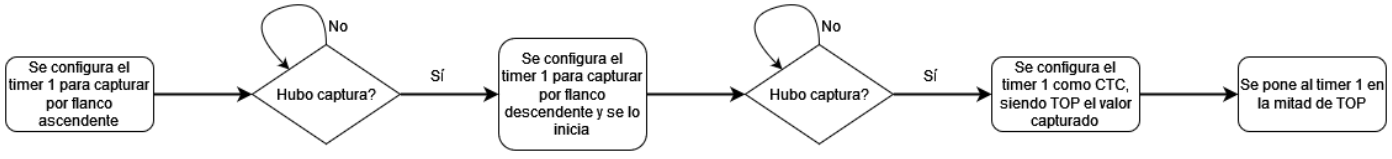

Figura 3: Diagrama del flujo de la obtención de la duración del punto.

Se configura el timer por última vez para funcionar en modo CTC y se empieza a leer el pin 0 del puerto B. Al recibir las letras en Morse, estas se procesan y van enviando a medida que se reciben.

Este proceso se repite indefinidamente. Hay que considerar que con velocidades de transmisión altas o mensajes muy largos puede llegar a haber una desincronización entre la lectura de los bits y la salida del sensor.

Recepción y envío por medio de la USART

Se utilizó comunicación asincrónica, sin bit de paridad, utilizando un bit de stop de largo 1, con un baude rate de 9600 bds.

Se utilizó un registro para almacenar el último carácter recibido o el valor que se quiere transmitir. Primero se espera a que el buffer de la USART esté disponible para leer o escribir y luego se almacena en este el valor del registro (o se almacena en el registro desde el buffer, si es una recepción).

Circuito

El circuito es está compuesto principalmente por las siguientes partes: el sensor de sonido, el microcontrolador con sus conexiones externas que están dentro del Arduino (botón de reset, clock de 16 MHz, ground), la conexión con el puerto serie y el sensor de sonido.

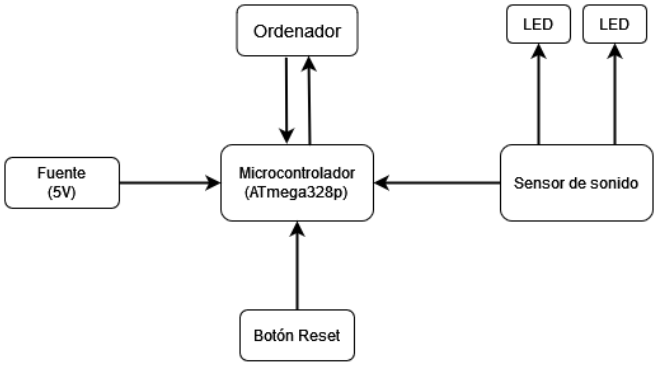

Figura 4: Diagrama en bloques del circuito.

Habiendo utilizado el pin analógico A0 del Arduino para poder leer bien la señal digital del sensor, se vio que había mucho ruido y que a veces el pin del puerto quedaba flotante. Razones de este puede haber sido el estado del protoboard utilizado, el hecho de no poder estar en un ambiente completamente silencioso o quizás la interferencia de los mismos cables.

En primer lugar se conectó un resistor de 100 kΩ entre el nodo que tiene al pin B0 y tierra. Este funcionó como resistor de pull-down, evitando que el puerto quede flotante. Luego se colocó un capacitor entre el mismo nodo y tierra para filtrar las frecuencias altas, disminuyendo el ruido que había mientras el bit estaba en high o low. Por último, se agregó un diodo entre la salida digital del sensor y B0, para que sumado al capacitor y al resistor funcione como detector de envolvente.

La forma de onda resultante fue la siguiente (en este ejemplo se ve el envío de “ET”):

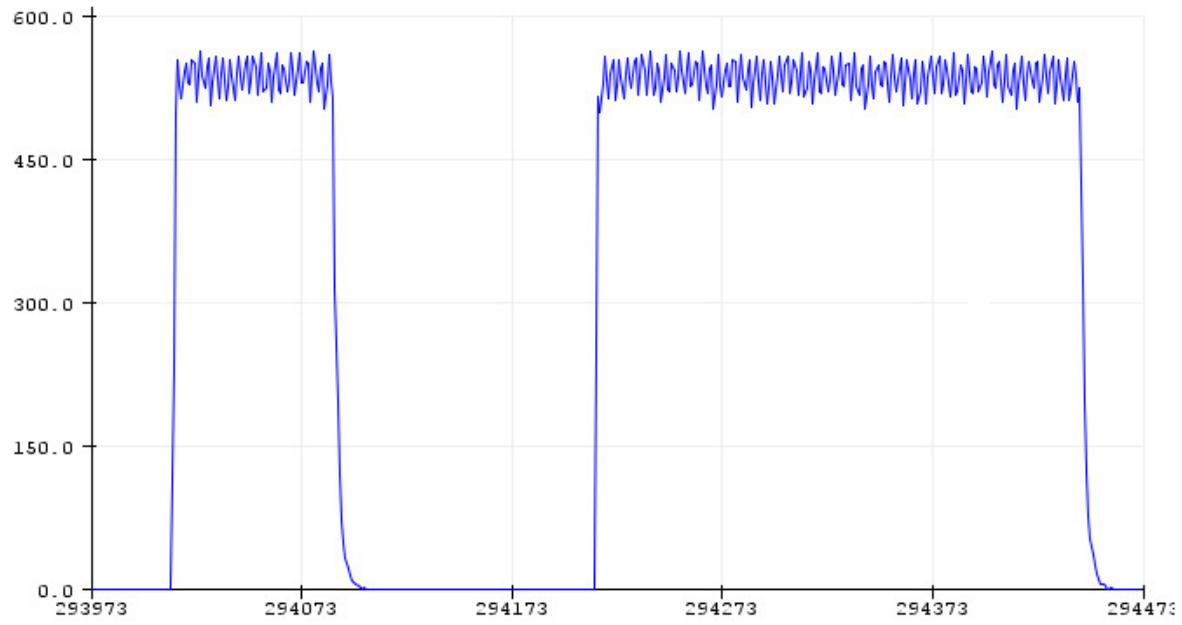

Figura 5: Forma de la onda leída con el la entrada analógica A0, graficada con la terminal de Arduino.

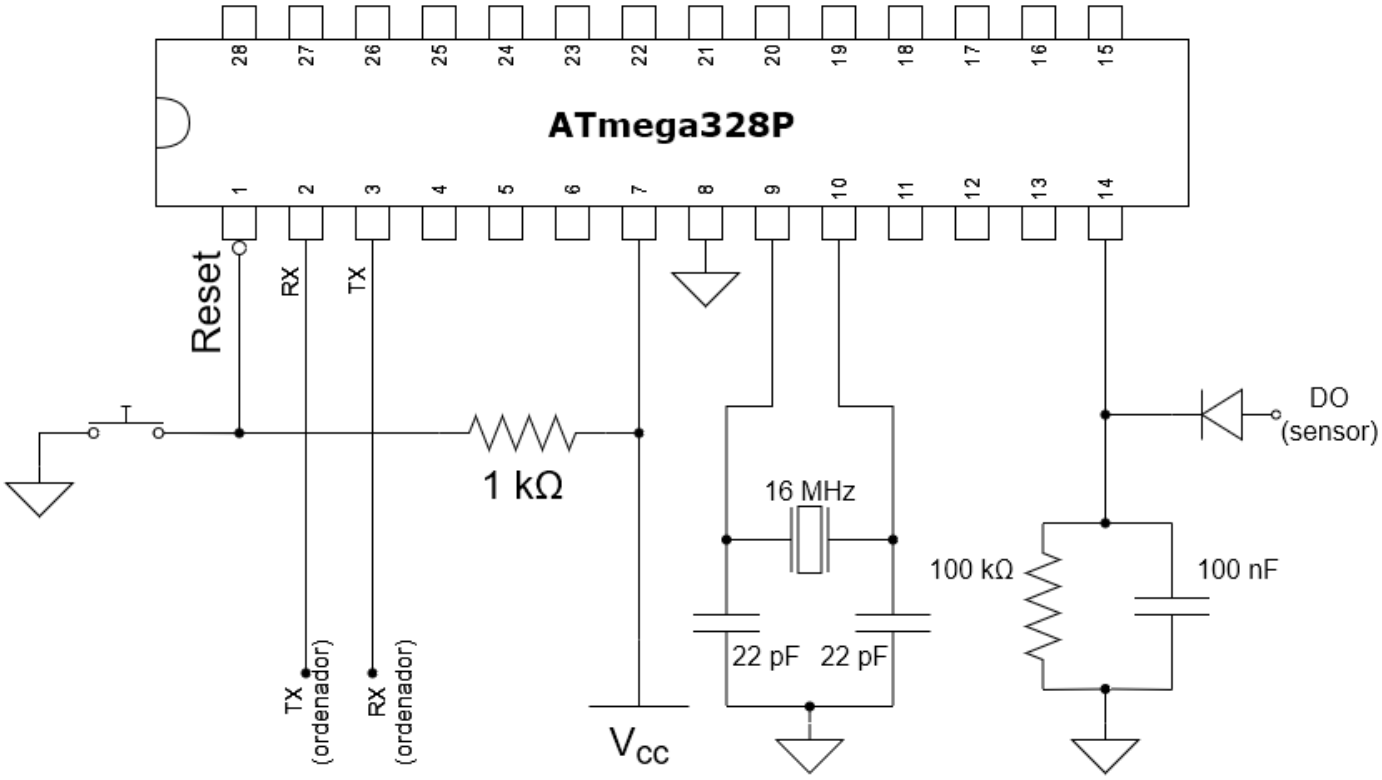

Figura 6: Diagrama esquemático del hardware.

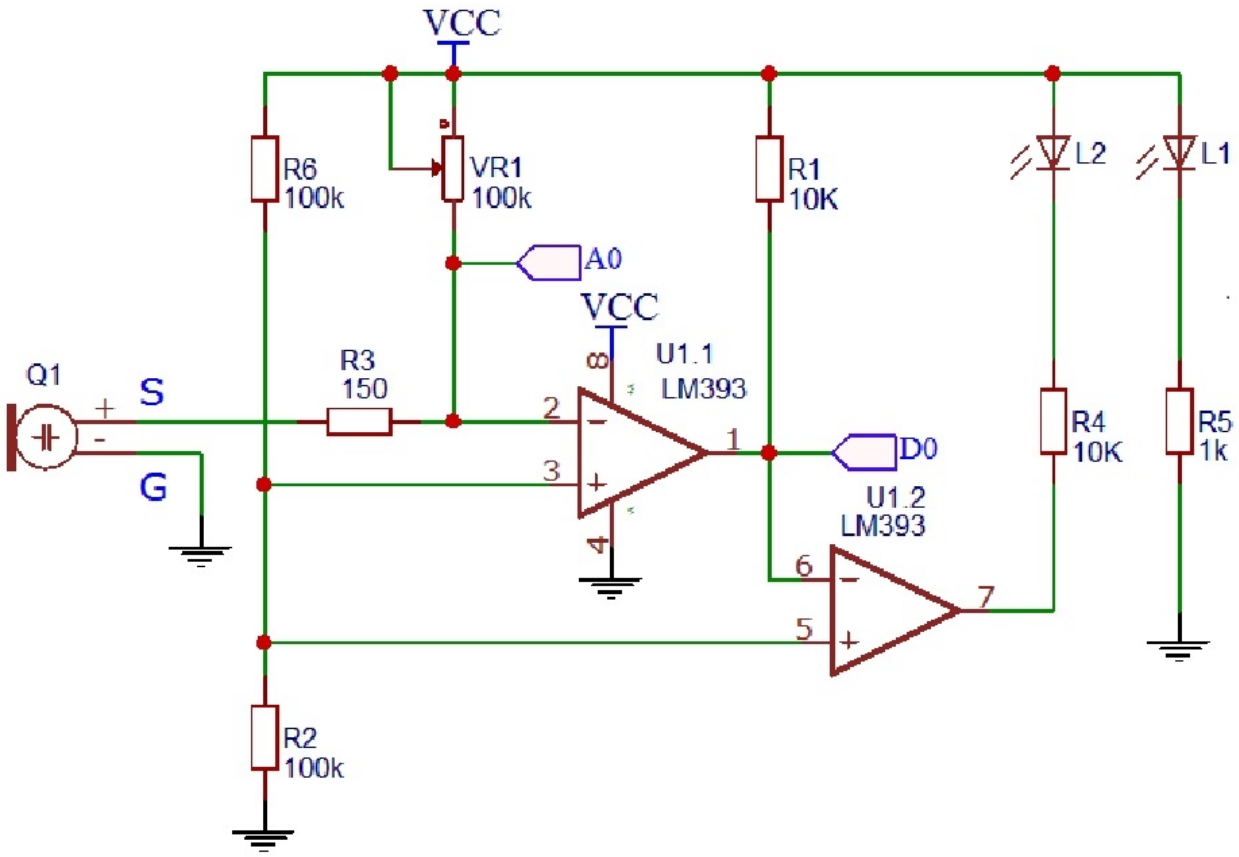

Figura 7: Diagrama esquemático del circuito interno del sensor de sonido.

Conclusiones

- Es importante conocer la forma de la onda que se va a procesar, de esta manera se podrá ver qué tipos de filtros harían falta y otras cosas a tener en cuenta como por ejemplo bouncing.
- Es buena idea utilizar LEDs si es que hay puertos disponibles para verificar el correcto funcionamiento del programa en el circuito. Una alternativa a esto es utilizar realizar envíos de información por la USART.
- La utilización de una secuencia conocida al momento de iniciar una transmisión ayuda mucho la obtención de su frecuencia.
- Es importante ir realizando el código y su circuito en conjunto, de tal manera que sea fácil detectar los nuevos problemas que vayan surgiendo.
- Al momento de probar el circuito es importante contar con un ambiente adecuado, en este caso uno sin muchos ruidos externos y en el cual se permita realizar ruido.
- Al momento de trabajar con comunicación asincrónica es importante que tanto receptor como emisor estén trabajando a la misma frecuencia y ambos sepan el largo de los mensajes.
- Es muy útil tener estandarizado un carácter para establecer el final de un string, como lo es el \0.