Universidad de Buenos Aires

Facultad de Ingeniería

Trabajo Práctico N° 2: Interrupciones externas, memoria EEPROM

ALUMNO

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

ASIGNATURA

86.07 - Laboratorio de Microprocesadores

23 de noviembre de 2022

Introducción y objetivos

En este trabajo se busca programar un controlador ATMega328P en Assembler y hacer su circuito para poder incrementar o decrementar el valor de un display de siete segmentos al presionar botones. Para el desarrollo del programa se utilizarán interrupciones y algunos valores serán guardados en la memoria EEPROM.

Lista de materiales

- 8 resistores de 220 Ω
- 1 display de cátodo común (C-391H)
- 1 microcontrolador ATmega328P
- 2 pulsadores
- 13 cables macho-macho para protoboard
- 1 protoboard
- 1 fuente de alimentación de 5V

Parte A

El software fue desarrollado con Microchip Studio, para luego ser ensamblado y subido al microprocesador con AVRDude. El programa se dividió en varios archivos: "main.asm", en el que está la estructura principal del programa; "7SegmentsDisplayDef.inc", en el que están las definiciones de los bits que se prenderán del display para representar cada número; "EEpromReadWriteSubroutines.asm", en el que están las rutinas de lectura y escritura de la memoria EEPROM; y "DisplaySubRoutines.asm", en el que se encuentran todas las subrutinas relacionadas con el encendido de los LEDs del display. Separar el código de esta manera permitió una mejor organización a lo largo del proyecto. Se programó para un display cátodo común, dado a que este fue el que se consiguió en la casa de electrónica.

El flujo del programa es el siguiente: primero se inician todos los puertos y configuran las interrupciones. Luego se muestra cada letra de la A a la F, dejando pasar 8.000.000 ciclos entre cada una y luego se muestra el valor inicial 0. A partir de este momento se habilitan las interrupciones y el programa queda a la espera de estas (habiendo llamado a `sleep`), en caso de presionar un botón ocurre una interrupción y se incrementa o decrementa un registro, luego usando máscaras se lo muestra por el display. En caso de querer decrementar a 0 o incrementar a F, no ocurre nada al presionar el botón. Este comportamiento se ve en el siguiente diagrama:

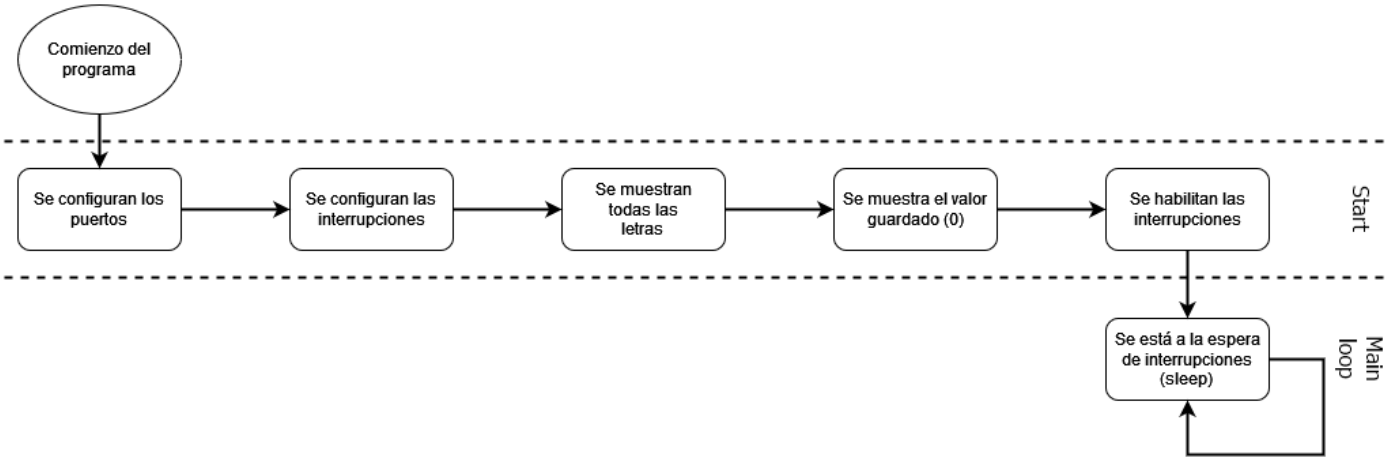

Figura 1: Diagrama del flujo del programa A hasta llamar a `sleep`.

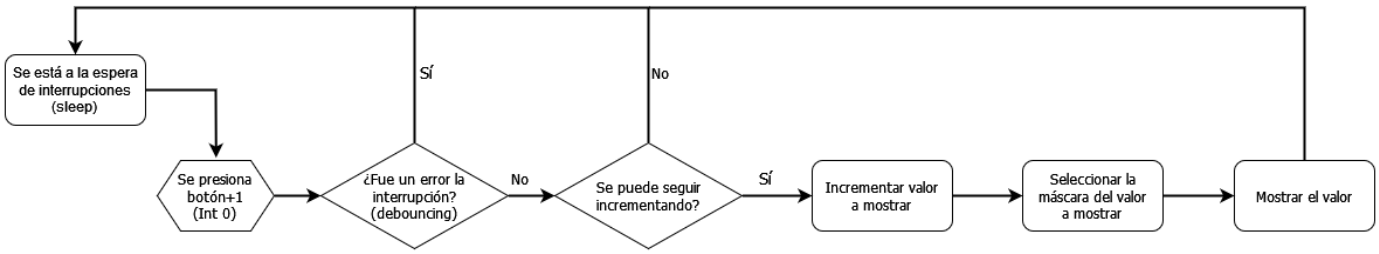

Figura 2: Diagrama del flujo del programa A cuando se llama a la interrupción de incrementar. La de decrementar es análoga.

Interrupciones:

En este programa fueron usadas para incrementar o decrementar el valor del display. Son llamadas al presionar un botón.

Las interrupciones utilizadas fueron programadas por flanco descendente, dado a que el botón está a V_{CC} cuando no está presionado y a GND cuando lo está. Estas se habilitan antes de estar a la espera de interrupciones por primera vez para que no se pueda incrementar/decrementar el valor del display mientras se está mostrando cada LED del display uno por uno.

Una vez que se llama a una interrupción se deja pasar un 1 ms, 16.000 ciclos de reloj, y luego se corrobora que efectivamente el botón haya sido presionado. Esto sirve para evitar tomar flancos causados por ruido y *bouncing*.

Luego se comprueba si el valor del display puede seguir siendo incrementado o decrementado y por último se incrementa/decrementa.

Display de números

Dado a que el ATmega328P está dentro de un Arduino nano, no todos los pines de los puertos estaban disponibles. El único puerto que tenía disponible todos sus pines disponibles era el "D", pero el D2 y el D3 se utilizaron para las interrupciones externas, y si se quisiese haber utilizado estos como output y como para interrupción al mismo tiempo, al presionar el botón que causaría la interrupción también se prendería un LED del display.

Por esto se optó por utilizar los primeros cuatro pines disponibles del puerto "C" y los últimos cuatro del puerto "D". Esta configuración permitió utilizar la misma máscara para mostrar el mismo número en ambos puertos, dado a que los bits de la máscara coincidían con el pin que debían establecer.

El guardado de las máscaras del 0 al 9 fueron hechas en un mapa en la memoria Flash del programa, mientras que para letras de la A a la F fueron guardadas en un mapa de memoria EEPROM. Dado a que el número que se debería mostrar está guardado en un registro, luego hay una subrutina encargada de relacionar la máscara con ese número. Para las letras, la subrutina también se encarga de leer la máscara de la EEPROM.

En caso que el número a mostrar no sea una cifra hexadecimal válida, se mostrará el número 0 para poder reanudar el programa sin problemas.

El diagrama en bloques del circuito planteado sería el siguiente:

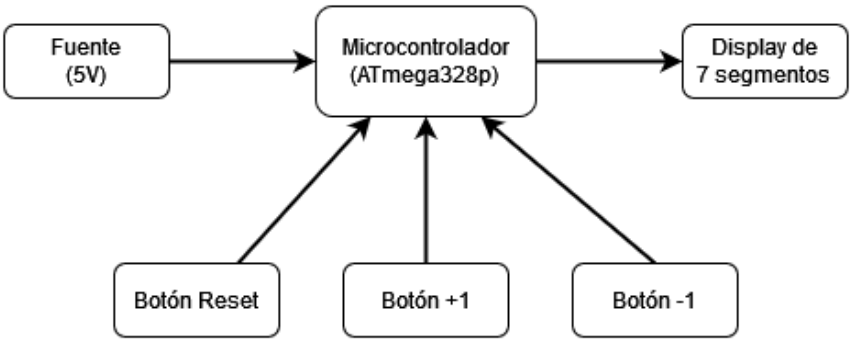

Figura 3: Diagrama en bloques del circuito A.

Parte B

La única diferencia entre esta parte y la anterior es que el último número mostrado debe ser guardado en la EEPROM y cuando se reinicie el programa este debe ser el primero mostrado.

Esto se hizo agregando en una posición de la EEPROM un valor que es leído y mostrado antes de esperar por las interrupciones durante el `sleep`. El valor es sobrescrito cada vez que se muestra un nuevo valor por el display. Dado a que esta es la única diferencia el diagrama de bloques y el esquemático del hardware serán iguales a los de la parte A y no se vuelven a mostrar.

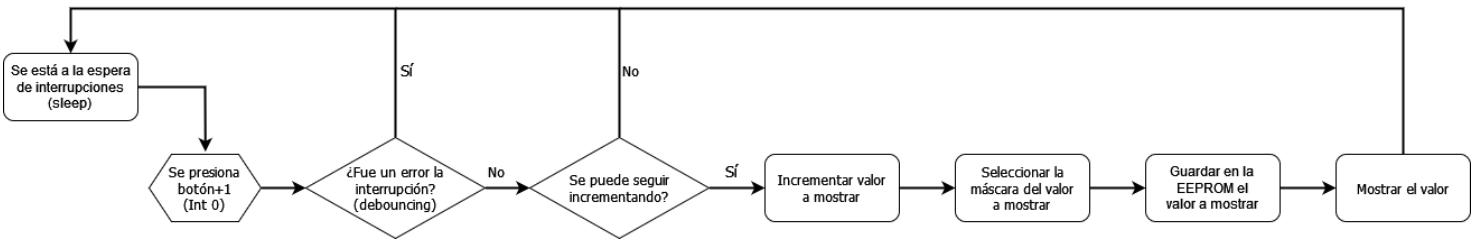

Figura 4: Diagrama del flujo del programa B.

Para el correcto funcionamiento, los dispositivos se deben conectar de la siguiente manera:

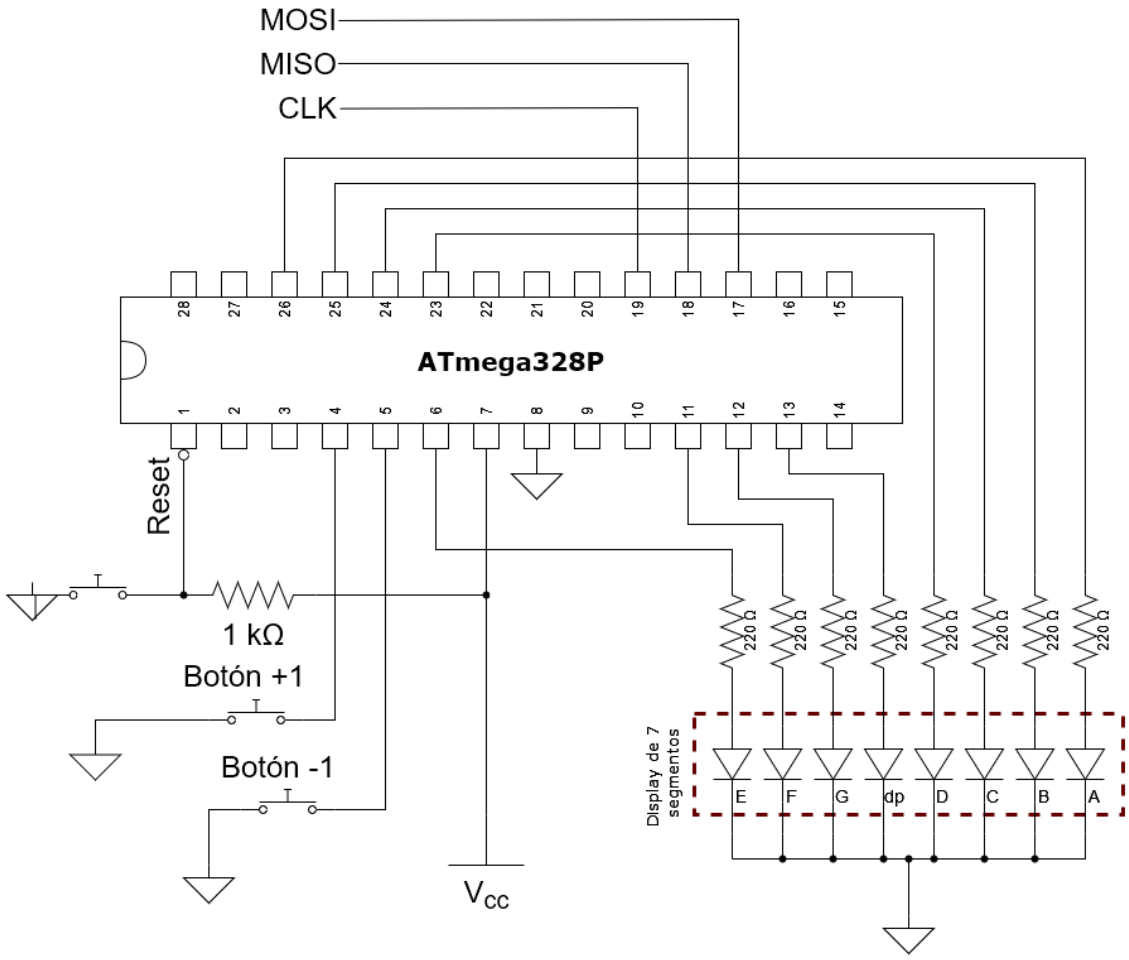

Figura 5: Diagrama esquemático del hardware.

Conclusiones

- Es importante programar las interrupciones teniendo en cuenta efectos como bouncing o errores, sino es muy probable que el circuito no funcione de manera esperada.
- Los pulsadores reales difieren bastante en funcionamiento a los pulsadores ideales, generando varios errores.
- La EEPROM resulta muy útil al momento de almacenar valores que no se desean perder al reiniciar el microcontrolador.
- En caso de haber utilizado toda o la mayoría de la memoria del programa, guardar información (como tablas de bytes) en la EEPROM puede ser una buena alternativa.
- AVR Studio no es una herramienta ideal al momento de simular programas a los cuales se les subió información a la EEPROM. En vez de hacerse directamente, hay que escribir durante la ejecución del programa a la EEPROM la información que se quiere tener al momento de simular, lo que puede resultar molesto.