Universidad de Buenos Aires
Facultad de Ingeniería
Año 2020 - 1^{ero} cuatrimestre

Laboratorio de Microprocesadores (86.07)

TRABAJO PRÁCTICO OBLIGATORIO 4

ESTUDIANTE:
Pintos Gaston 99711
gmpintos@fi.uba.ar

Índice

1. Objetivos

El trabajo práctico consiste en generar un programa que sea capaz de detectar una interrupción generada por un pulsador. El programa deberá encender un LED y esperar al accionamiento de la interrupción. Cuando se genere la interrupción se deberá apagar el primer LED y generar el parpadeo de un segundo LED a una determinada frecuencia. Se utilizo el microcontrolador *atmega 328P* de la placa *Arduino*. A lo largo del desarrollo se presentaran las herramientas utilizadas para la realización del proyecto.

2. Desarrollo:

2.1. Herramientas de Software:

Para el siguiente proyecto se utilizo la herramienta *AtmelStudio* para desarrollar el código en lenguaje *Assembly*. La misma herramienta permite simular el código escrito con lo cual fue posible realizar pruebas a lo largo del desarrollo. Además se utilizo la herramienta *AVRdude* para poder montar el programa en el microcontrolador.

2.1.1. Código de Montaje:

Se utilizo el siguiente código para cumplir la tarea asignada:

```
    .device ATmega328P
; Vector de Main
.org 0
                rjmp Main
; Vector de INT0 (pin PD2)
.org INT0addr
                rjmp IntV0

; Inicializa stack
Main:   ldi R16,low(RAMEND)
                out SPL,R16
        ldi R16, high(RAMEND)
        out SPH, R16

                call INIT_IRQ_INT0

                clr R16
                sbi DDRB, 0 ; PORTB pin8 como salida
                sbi DDRB, 1 ; PORTB pin9 como salida
                cbi DDRD, 2 ; PORTD pin4 como entrada
                ;sbi PORTD, 2 ; PORTD RESISTENCIA PULLUP
                sei                 ; Hab. global de interrupciones
                sbi PORTB,0 ; Enciendo el LED DEL pin 8
loop:
                rjmp loop       ;GENERO UN LOOP PARA ESPERAR LA INTERRUPCION


IntV0:
        CALL DELAY
        sbis PORTD,2
        call INTERVALO
        reti

INTERVALO:
        cbi PORTB,0
        call ENCEDER_LED_INT
        sbi PORTB,0
        ret

ENCEDER_LED_INT:
```

```
        LDI     R22,5
LOP_1:
        sbi PORTB,1
        CALL DELAY
        cbi PORTB,1
        CALL DELAY
        DEC R22
        BRNE LOP_1
        RET

DELAY:
    ldi  r18, 41
    ldi  r19, 150
    ldi  r20, 128
L1: dec  r20
    brne L1
    dec  r19
    brne L1
    dec  r18
    brne L1
        ret

; Configura INT0 con el flanco ascendente
INIT_IRQ_INT0:
                ldi R16, (1<<ISC01)|(1<<ISC00)   ;flanco ascendente
                sts EICRA, R16
                ldi R16, (1<<INT0)
                out EIMSK, R16
                RET
```

Se utilizo la instrucción *.DEVICE* para identificar el tipo de microcontrolador utilizado. Con la directiva *.ORG* se seteo el *stack pointer* al inicio de la memoria flash donde se ubican las directivas del programa. Con esta directiva se indican dos sectores de código distinto referidos al programa principal *Main* y la interrupción *IntV0* utilizando el pin INT0 especifico para la generación de interrupciones.

La primera tarea del *Main* es inicializar el Stack. Luego se llama a la función *INIT_IRQ_INTO*. Esta función se encarga de configurar el pin INT0 utilizando el registro de control de interrupción EICRA para asociar la interrupción con el flanco ascendente y se habilita la interrupción externa para INT0 con el registro EIMSK. Después se configuran el puerto D como entrada y el puerto B como salida. Se utiliza la instrucción *sei* para habilitar las interrupciones globales, se enciende el LED conectado al pin 8 del puerto B, y por ultimo se entra a un loop infinito cuya razón es esperar a que se produzca alguna interrupción externa.

Una vez generada la interrupción, el programa ingresa en la función *IntV0*. Dentro de esta función se encuentra un DELAY para evitar el efecto rebote que pueda generar el pulsador. La instrucción *sbis* verifica que haya sido efectiva la activación de la interrupción descartando algún error de contacto. Ante esta verificación, se ingresa en la función INTERVALO cuya función sera apagar el LED 0, generar el parpadeo del LED 1 y volver a encender el LED 0. Una vez finalizada la tarea de la interrupción esta ejecuta la instrucción *reti* para regresar a la función principal.
El parpadeo del LED 1 conectado al pin 9 del puerto B se genera mediante un loop que utiliza el registro R22 como contador y la instrucción BRNE con condición. Entre los sucesivos encendido-apagdo del LED se utilizo una rutina de DELAY para cumplir con la especificación de la frecuencia.

2.1.2. Calculo del retardo:

Para el siguiente código se utilizo el reatrdo para generar 5 parpadeos con una frecuencia de 1Hz o bien que prende durante $0,5s$ y se apague durante $0,5s$ para una frecuencia de clock característica de $16MHz$:

```
    DELAY:
    ldi  r18, 41        1 ciclo
    ldi  r19, 150       1 ciclo
    ldi  r20, 128       1 ciclo
```

```
L1: dec   r20            1 ciclo
    brne L1              1 ciclo si no cumple/ 2 si cumple
    dec   r19            1 ciclo
    brne L1              1 ciclo si no cumple/ 2 si cumple
    dec   r18            1 ciclo
    brne L1              1 ciclo si no cumple/ 2 si cumple
        ret
```

$$tiempo = \frac{ciclos}{frecuencia} = \frac{((128 * 3) - 1 + 3) + 149 * ((256 * 3) - 1 + 3) + 41 * 256 * ((256 * 3) - 1 + 3)}{16000000} = 0{,}51s \quad (1)$$

2.1.3. Montaje del código:

Paecuencia dera el montaje del código con la herramienta *AVR DUDE* a través del *ATMEL STUDIO* se utilizaron los siguientes comandos:
"C:9-Laboratorio de Microcomputadoras-6.3-mingw32(1).exe"-C
"C:9-Laboratorio de Microcomputadoras-6.3-mingw32(1).conf" -v
-p atmega328p -c arduino -P COM3 -D
-U flash:w:"$(MSBuildProjectDirectory)$(Configuration)$(OutputFileName).hex":i

Con esta herramienta fue posible que el código escrito fuera procesado por el microcontrolador para poder ejecutar las tareas programadas.

2.1.4. Diagramas de flujo:

Para una comprensión de manera gráfica se presenta a continuación un diagrama de flujo del código previamente explicado.

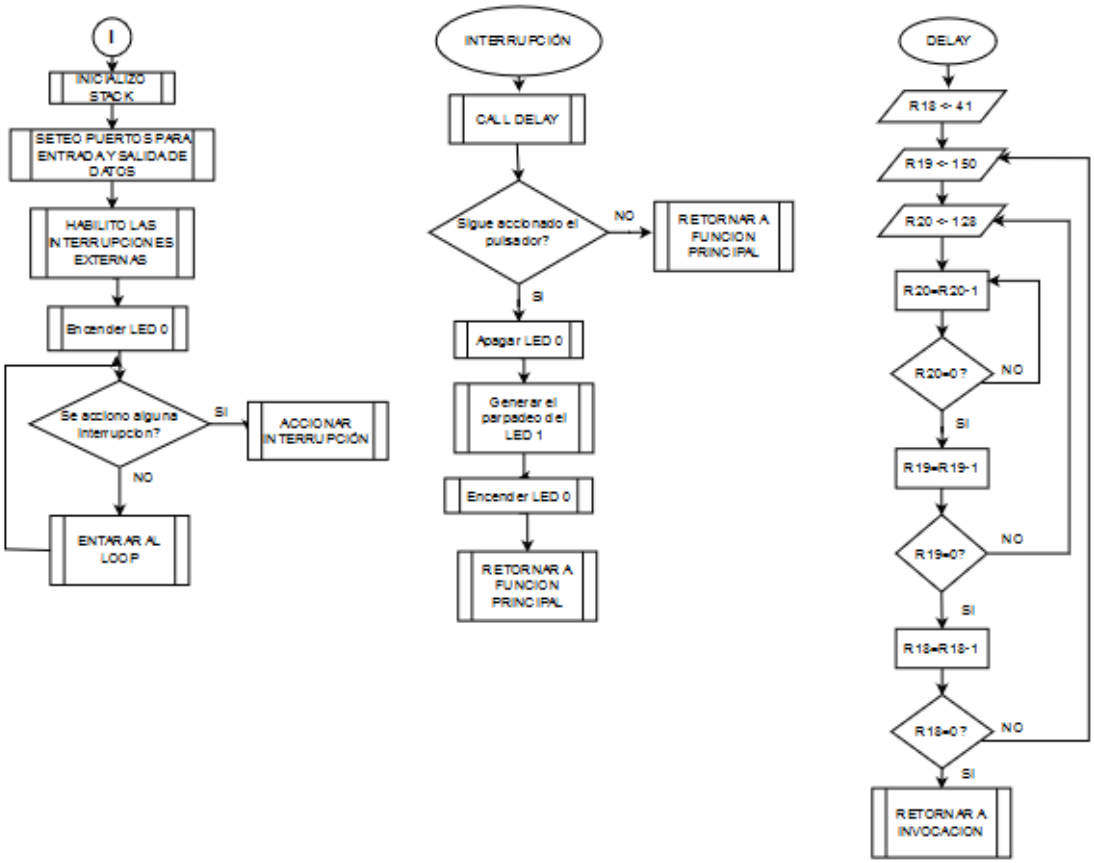

Figura 1: Diagrama de flujo de código.

2.2. Herramientas de Hardware:

Para realizar el proyecto se utilizaron el microcontrolador de la placa *Arduino Uno*, un protoboard donde se montaron todos los elementos, 2 diodos led, un pulsador, una resistencia de $10K\Omega$ y 2 resistencias de 220Ω. Para la alimentación y trasmisión del código se utilizo el puerto USB que viene con la placa. Para interconexión entre la placa y el protoboard se utilizaron cables macho-macho. Se puede visualizar la conexión de los elementos en la siguiente imagen.

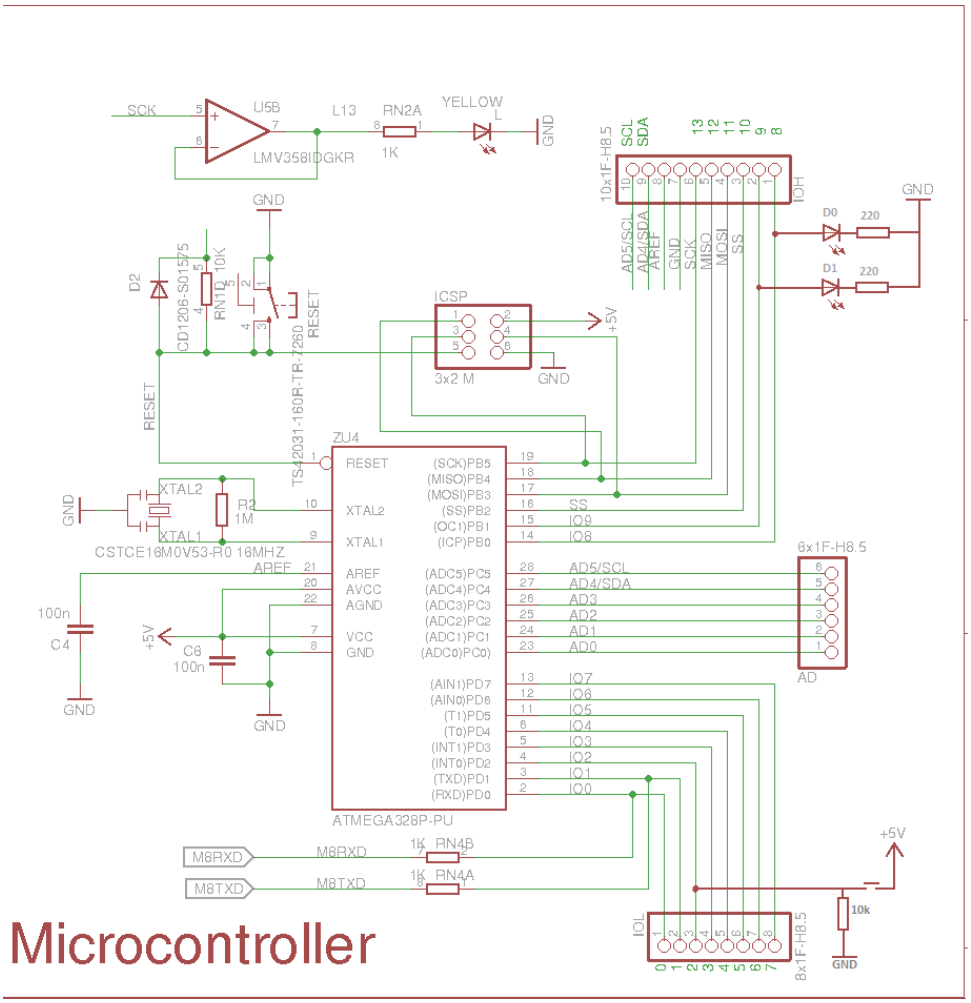

Figura 2: Diagrama de conexión.

La placa *Arduino Uno* cuenta con un microcontrolador *ATmega328p* . El ATmega328 proporciona comunicación serie UART TTL (5V), que está disponible en los pines digitales 0 (RX) y 1 (TX), Dentro de sus conexiones cuenta con 14 pines digitales de entrada/salida. Tiene una memoria flash de 32KB, una SRAM de 2KB y una EEPROM de 1KB. La velocidad del reloj es de 16MHz.

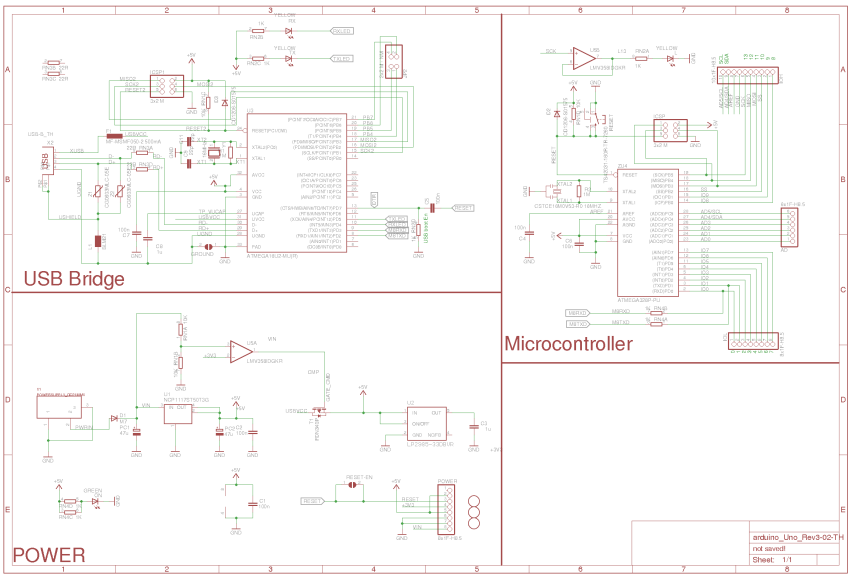

Figura 3: Diagrama de conexión de la placa.

2.2.1. Listado de componentes

Para este trabajo practico fueron utilizados los siguientes elementos. Se estima un coste total de 1300$.

Artículo	Precio
Arduino Uno	$800 - 1500$$
Led	$20 - 30$$
Resistencia	$5 - 10$$
Pulsador	$10 - 20$

Tabla 1: Precios de componentes.

2.3. Resultados

En el momento de realizar el código se busco poder desarrollar el mismo con el objetivo de generar interrupciones externas. Estas logran que el programa realice una tarea especifica si la interrupción es accionada. Cuando ya se realiza la tarea el programa sigue ejecutando el código utilizado hasta la interrupción. Para esto fue necesario habilitar la interrupciones globales e introducir una función especifica que configura las mismas.
Se implemento el código utilizando la instrucción *call* en cada llamado a la función encargada de desplazar el bit según fuera necesario. Se evito utilizar la instrucción*rjmp* en la llamada a funciones para evitar problemas en el stack.
Se utilizo la instrucción *rjmp* como indicador de la ubicación del programa principal como de la interrupción al inicio del código.

2.4. Conexión de la resistencia PullUp:

Los microcontroladores AVR cuentan con una resistencia pull-up conectada en cada pin. Esta evita una entrada flotante en la conmutación de estados. Además sirve para definir los estados ante la conexión de grandes impedancias. Esta resistencia se activa en cuanto se setean los pines del puerto en estado alto. Podemos observar la disposición en la siguiente imagen:

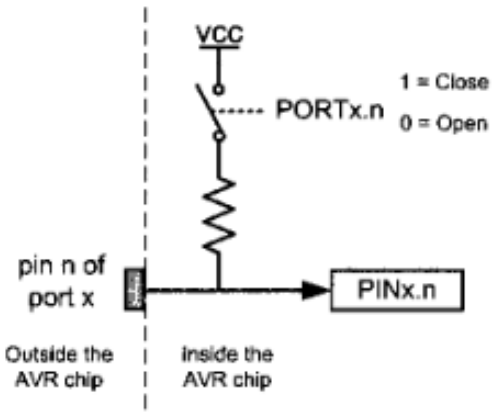

Figura 4: Diagrama de conexión de la placa.

En nuestro diseño, se podría prescindir de las resistencias series conectadas al pulsador ya que se logra alcanzar los estados high y low por el uso de la resistencia pull-up. Sin embargo, el diseño del programa debería ser modificado ya que al estar conectado a tierra el pulsador este mismo ingresa un 0 lógico. El siguiente código se utilizo para implementar esta herramienta:

```
.device ATmega328P
; Vector de Main
.org 0
            rjmp Main
; Vector de INT0 (pin PD2)
.org INT0addr
            rjmp IntV0

; Inicializa stack
Main:       ldi R16,low(RAMEND)
            out SPL,R16
        ldi R16, high(RAMEND)
        out SPH, R16

            call INIT_IRQ_INT0

            clr R16
            sbi DDRB, 0 ; PORTB pin8 como salida
            sbi DDRB, 1 ; PORTB pin9 como salida
            cbi DDRD, 2 ; PORTD pin4 como entrada

            ldi R17, 0xFF
            out PORTD,R17 ;Habilito resistencia PullUp

            sbi PORTB,0 ; Enciendo el LED DEL pin 8

            sei                 ; Hab. global de interrupciones

loop:
            rjmp loop         ;GENERO UN LOOP PARA ESPERAR LA INTERRUPCION


IntV0:
    CALL DELAY
    sbic PORTD,2
    call INTERVALO
    reti
```

```
INTERVALO:
    cbi PORTB,0
    call ENCEDER_LED_INT
    sbi PORTB,0
    ret

ENCEDER_LED_INT:
    LDI     R22,5
LOP_1:
    sbi PORTB,1
    CALL DELAY
    cbi PORTB,1
    CALL DELAY
    DEC R22
    BRNE LOP_1
    RET

DELAY:
    ldi  r18, 41
    ldi  r19, 150
    ldi  r20, 128
L1: dec  r20
    brne L1
    dec  r19
    brne L1
    dec  r18
    brne L1
    ret

; Configura INT0 con el flanco ascendente
INIT_IRQ_INT0:
            ldi R16, (1<<ISC01)|(1<<ISC00)   ;flanco ascendente
            sts EICRA, R16
            ldi R16, (1<<INT0)
            out EIMSK, R16
            RET
```

Se implementa el siguiente circuito:

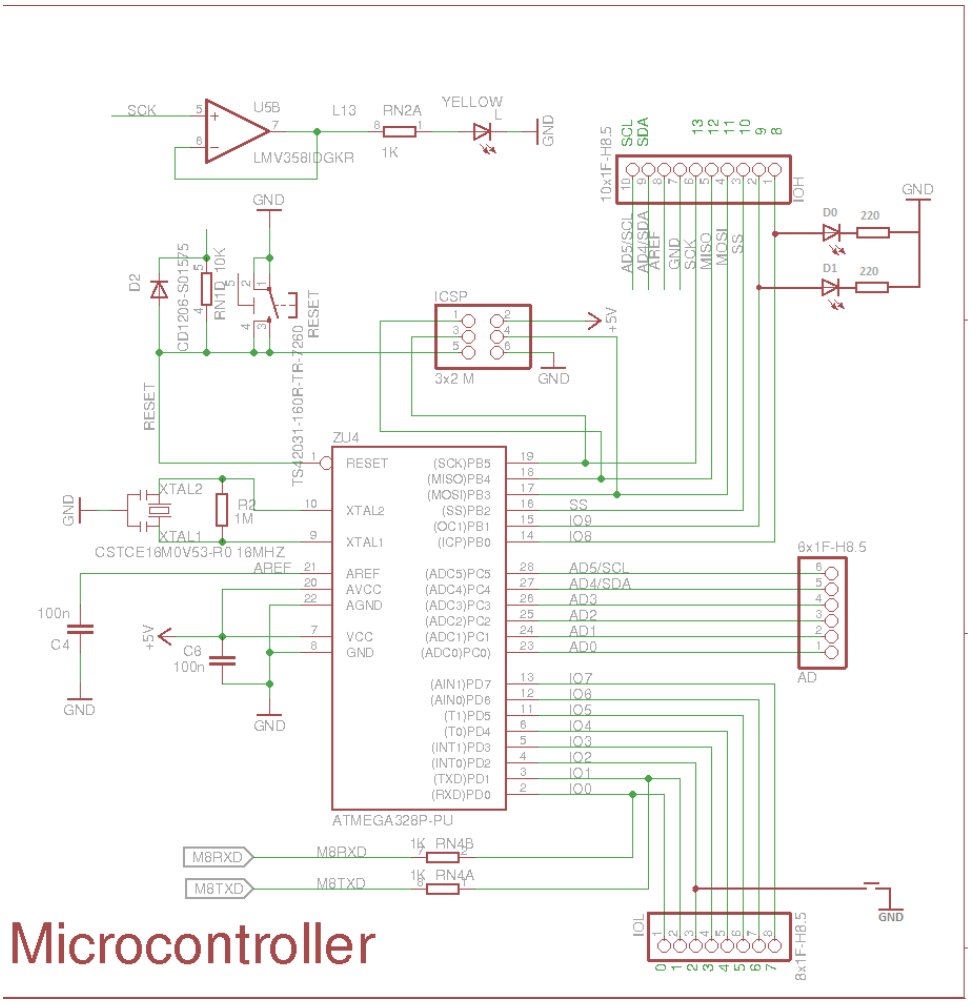

Figura 5: circuito implementado para la resistencia pull-up.

3. Conclusiones

Para este trabajo practico se introdujeron las interrupciones como un método eficiente ante las realización de diversas tares por parte del microcontrolador. Se configuraron utilizando los registros EICRA y EIMSK para poder obtener respuestas ante un flanco ascendente. EL encendido correcto de LED ante una interrupción generada por el pulsador pudo verificar el cumplimiento de la consigna.

4. Bibliografía

- AVR Microcontroller and Embedded Systems: Using Assembly and C, 1stEdition, Muhammad Ali Mazidi, Pearson
- "Digital Design, Principles and Practices", 3rdEdition, J.Wakerly, Prentice-Hall
- Set de instrucciones AVR de 8 bits
- Execution cycle of the AVR architecture
- Guía de Trabajos Prácticos
- ATmega328P Datasheet