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

Laboratorio de Microprocesadores (86.07)

TRABAJO PRÁCTICO OBLIGATORIO 2

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

Índice

1. Objetivos

El trabajo practico consiste en utilizar el lenguaje de programación *Assembly* para llevar a cabo del encendido de un led a través de instrucciones al microcontrolador. Este encendido sera controlado mediante un pulsador conectado a un puerto de entrada y provocara el encendido de un led ubicado en un puerto de salida.Para este trabajo 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. Codigo de Montaje:

Para la tarea se conecto el pulsado el puerto D en la posición seteado como pin de entrada y el led fue ubicado en el puerto B en la posición seteado el mismo como pin de salida. Se utilizo el siguiente código:

```
.device ATmega328P
.cseg
.org 0x0000

.EQU PIN_ENTRADA=2
.EQU PIN_SALIDA=4

main:

    LDI R16,HIGH(RAMEND)
    OUT SPH,R16
    LDI R16, LOW(RAMEND)
    OUT SPL, R16

    sbi DDRB, PIN_SALIDA
    cbi DDRD, PIN_ENTRADA

CHEQUEAR_APAGADO:
    sbis PIND, PIN_ENTRADA
    call APAGAR_LED
    rjmp CHEQUEAR_ENCENDIDO

CHEQUEAR_ENCENDIDO:
    sbic PIND, PIN_ENTRADA
    call ENCENDER_LED
    rjmp CHEQUEAR_APAGADO

ENCENDER_LED:
    sbi PORTB, PIN_SALIDA
    ret

APAGAR_LED:
    cbi PORTB, PIN_SALIDA
    ret
```

Se utilizo la directiva *.DEVICE* para identificar el tipo de microcontrolador utilizado. La directiva *.CSEG* indica la locación de memoria utilizada para escribir el código. Con la directiva *.ORG* se seteo el *stack pointer* al inicio de la memoria flash donde se ubican las directivas del programa. Con la directiva *.EQU* se definieron las variables "$PIN\_ENTRADA$" y "PIN_SALIDA".Al definirlas de esta manera es posible modificar los pines de los puertos sin necesidad de modificar el código.

La primera tarea realzada por el main es configurar los puertos. Para esto se introdujo el valor 1 en la posición correspondiente del "PIN_SALIDA" al DDRB lo cual lo instala como pin para la salida de datos, y se introdujo el valor 0 en la posición correspondiente del "$PIN\_ENTRADA$" al DDRD lo cual lo instala como pin para la entrada de datos.
El programa luego ejecuta la función "$CHEQUEAR\_APAGADO$". Esta comienza con la instrucción **sbis** la cual verifica si el pin de entrada esta en 1. En caso de estarlo la función llama a "$CHEQUEAR\_ENCENDIDO$". En caso de no estarlo la función llama a "$APAGAR\_LED$". Como caso análogo, la función "$CHEQUEAR\_ENCENDIDO$" verifica el pin de entrada con la instrucción **sbic** que saltea la próxima instrucción si el pin de entrada esta en 0. En ese caso llama a la funcion "$CHEQUEAR\_APAGADO$", caso contrario llama a "$ENCENDER\_LED$".
La funcion de "$ENCENDER\_LED$" introduce el valor 1 en el pin de salida con intruccion **sbi**.La funcion de "$APAGAR\_LED$" introduce el valor 0 en el pin de salida con intruccion **cbi**.

Todas las instrucciones utilizadas para la escritura del código se encuentran en el manual de instrucciones del microcontrolador. El mismo puede encontrarse en la bibliografía al finalizar el trabajo.

2.1.2. Calculo del retardo:

Para el siguiente código no se utilizo ningún tipo de *delay* debido a que durante su implementación física no se observo ningún tipo de rebote. El rebote mencionado es un tipo de imperfección en la transición de 0v a 5v que experimenta una entrada, cuando tiene conectada un pulsador que conmuta estos estados. Sin embargo no se visualizo tal imperfección. La solución hubiera es introducir un *delay* luego de cada llamada a las funciones encargadas de encender y apagar el led.

2.1.3. Montaje del código:

Para 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 codigo 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.

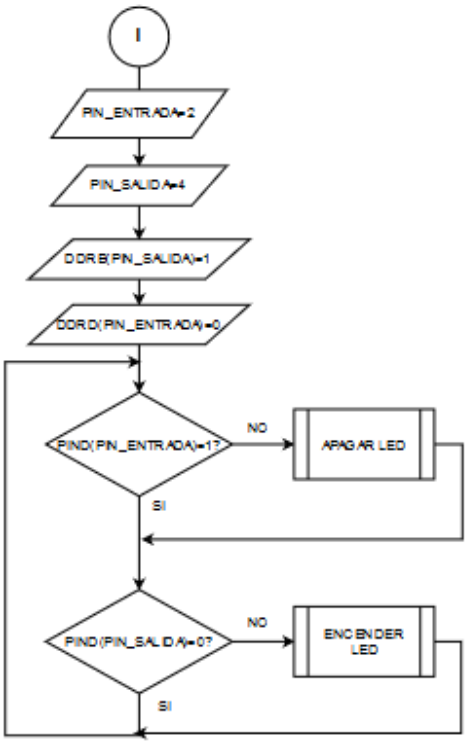

Figura 1: Diagrama de flujo de código.

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, un led, un pulsador, dos resistencias de $4K7\Omega$ y una resistencia de 260Ω. 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.

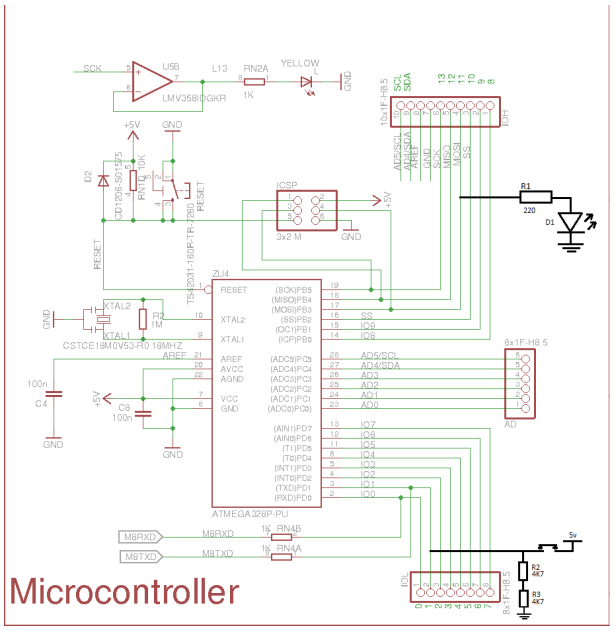

Figura 2: Diagrama de conexión.

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.

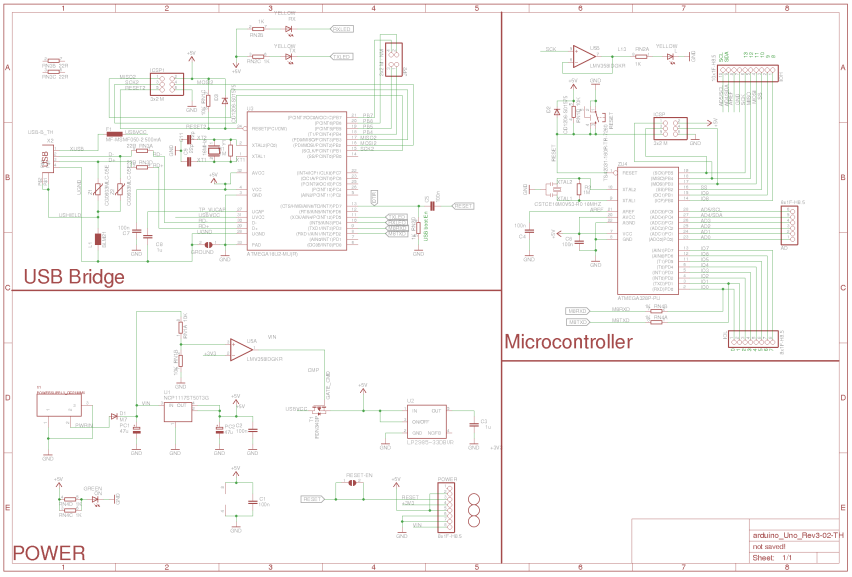

Figura 3: Diagrama de conexión de la placa.

2.2.1. Listado de componentes

Para este trabajo practico fueron utilizados los siguientes elementos. Se estima un valor total de 1250$

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

Tabla 1: Precios de componentes.

2.3. Resultados

En el momento de realizar el código se busco que pueda identificar los distintos estados del pulsador. Bajo este requerimiento fue necesario introducir dos nuevas instrucciones como *sbic* y *sbis* las cuales son ideales para verificar esta condición.
Se implemento el código utilizando la instrucción *call* en cada llamado a la función de encendido y apagado con el fin de obtener un correcto funcionamiento. Se evito utilizar la instrucción*rjmp* en la llamada a funciones para evitar problemas en el stack.
Se utilizo la instrucción *rjmp* como consecuencia lógica al verificar la condición del pin de entrada pero no se ejecuto un *ret* al finalizar los mismos para evitar un problema de stack.
Se introdujeron directivas al compilador para poder informar el dispositivo utilizado, el segmento de memoria, definir variables y setear el stack pointer.

2.4. Resistencia Pull-up:

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:

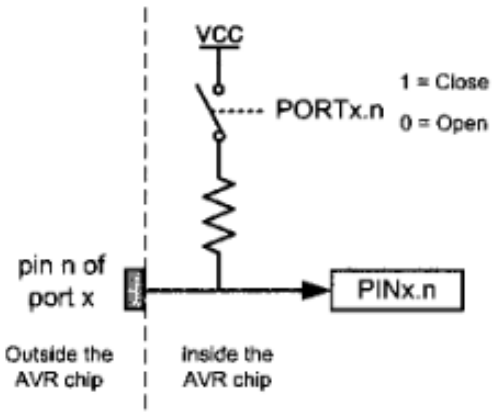

Figura 4: Diagrama de conexión de la placa.

En nuestro diseño, si queremos implementar el uso de la resistencia pull-up se debería setear en 1 los valores del puerto D. Con esto 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
.cseg
.org 0x0000

.EQU PIN_ENTRADA=2
.EQU PIN_SALIDA=4
```

```
main:

    LDI R16,HIGH(RAMEND)
    OUT SPH,R16
    LDI R16, LOW(RAMEND)
    OUT SPL, R16

    sbi DDRB, PIN_SALIDA
    LDI R17, 0XFF
    OUT PORTD,R17

CHEQUEAR_APAGADO:
    sbis PIND, PIN_ENTRADA
    call APAGAR_LED
    rjmp CHEQUEAR_ENCENDIDO

CHEQUEAR_ENCENDIDO:
    sbic PIND, PIN_ENTRADA
    call ENCENDER_LED
    rjmp CHEQUEAR_APAGADO

ENCENDER_LED:
    sbi PORTB, PIN_SALIDA
    ret

APAGAR_LED:
    cbi PORTB, PIN_SALIDA
    ret
```

Se implementa el siguiente circuito:

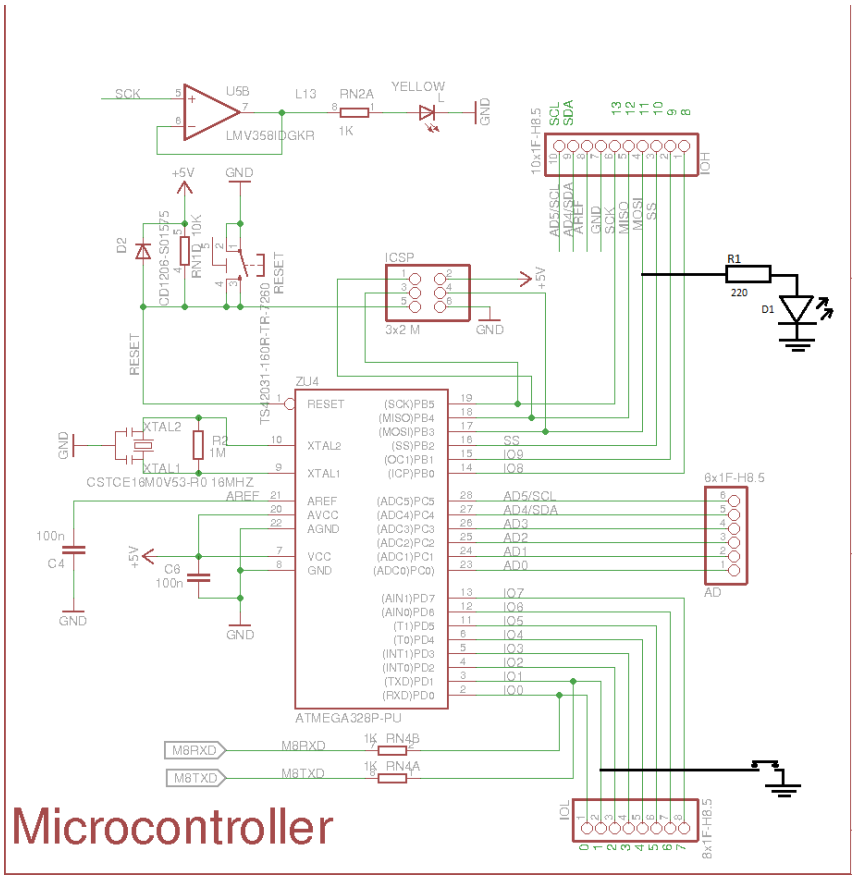

Figura 5: circuito implementado para la resistencia pull-up.

3. Conclusiones

Para este trabajo practico se pudieron comunicar distintos puertos del microcontrolador para cumplir el objetivo del trabajo. Se desarrollo un código a fin que pudo corroborar los estados de la entrada y a partir de esto discernir los valores de salida. La implementación física del diseño del programa genero una mayor comprensión del uso y la implementación de los puertos de salida y entrada que posee un microcontrolador..

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