Laboratorio de microcomputadoras 66.09
Laboratorio de microprocesadores 86.07

Guía de Trabajos Prácticos

1° cuatrimestre

Año 2020

INDICE

Introducción

Antes de empezar a leer esta guía, es recomendable leer la guía de usuario del programador, en la que se explica cómo usar el Programador y como instalar los programas necesarios para poder empezar a programar.

Para realizar estas prácticas deberán conseguir algunos materiales.

Las prácticas pueden hacerse con cualquier microcontrolador de la familia ATmega, en esta guía están explicadas para el microcontrolador ATmega328 o compatible, de 28 pines. Si usan otro micro, por ejemplo si usan un ATmega32 que tiene 40 pines, deberán ver a que pines tienen que conectar los elementos, leds, resistencias, etc.

En esta guía nos referimos también a **micro** en forma abreviada haciendo referencia al microcontrolador.

Pueden usar la placa de desarrollo Arduino que contiene un micro

ATmega 328p, deben ver que pines del conector del arduino conectan con los pines del micro pedidos en la práctica.

Al final de esta guía se adjunta imagen de pinouts de varios micros y arduino uno.

Trabajo Práctico N° 1
-Parpadeo de un LED

Objetivo:

Manejar el puerto para hacer parpadear un LED

Descripción

Realicen un programa que haga parpadear al LED conectado al pin (PB0 según imagen). Se pide realizar 2 programas donde el parpadeo se hará utilizando dos métodos distintos de programación:

-utilizando todo el puerto

-utilizando solo un bit del puerto, que corresponde el pin al que le conectamos el LED que parpadeará.

Materiales

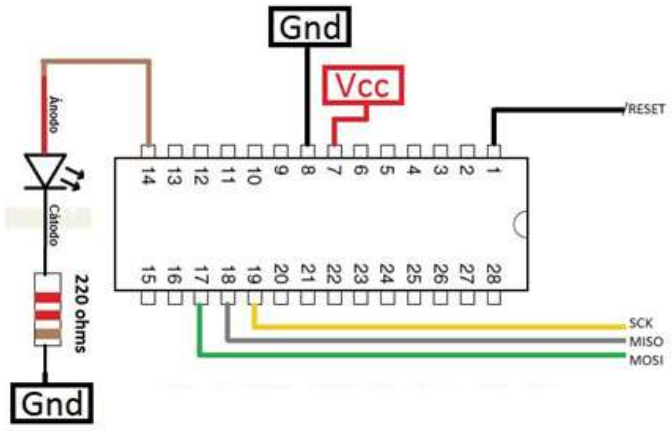
1 LED

1 Resistencia de 220 Ohms

1 Microcontrolador ATmega328p

Programador USBasp V3.0 (según disponibilidad)

Diagrama Esquemático



Capítulos de lectura recomendados

"The AVR microcontroller and embedded systems. Embedded system using Assembly and C". Autores: MUHAMMAD ALI MAZIDI, SARMAD NAIMI, SEPEHR NAIMI

Capítulo 4

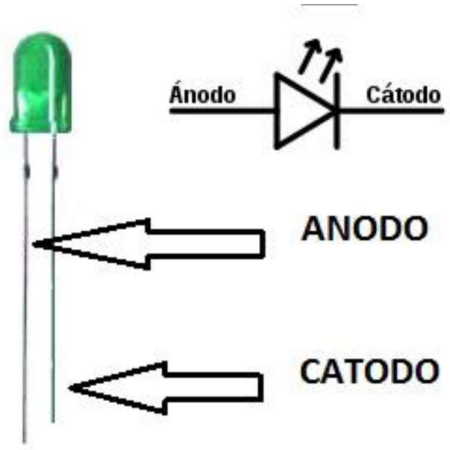
Resumen teórico

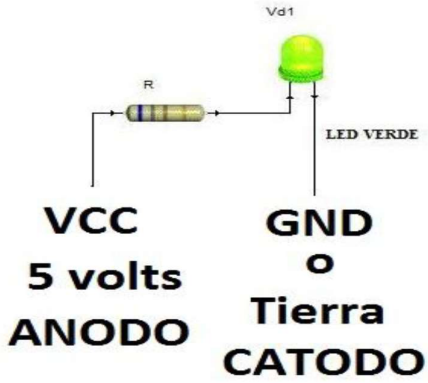
LED

LED acrónimo de Light Emitting Diode o Diodo Emisor de Luz, es un dispositivo semiconductor que emite luz al circular a través de él una corriente eléctrica.

Los LEDs como los diodos normales tienen un ánodo y un cátodo, los cuales se pueden identificar de manera fácil, siendo el ánodo la pata más larga como se observa en la figura.

Parpadeo de un LED





Como se puede apreciar en la figura superior, al LED se le coloca una resistencia en serie para limitar la corriente del mismo, en este caso que se alimentará con 5v se sugieren usar resistencias de 220 Ohms.

Para calcular las resistencias exactas y obtener un desempeño óptimo se realiza a través de la siguiente fórmula:

$$R = \frac{Vcc - V_{LED}}{I_{LED}}$$

Color	Caida de tensión (VLED) V	Intensidad máxima (ILED) mA	Intensidad media (ILED)mA
Rojo	1.6	20	5 – 10
Verde	2.4	20	5 – 10
Amarillo	2.4	20	5 – 10
Naranja	1.7	20	5 – 10

Caida de tensión e intensidad.

Tensión de alimentación,VCC , es el voltaje aplicado al circuito

Caída de tensión del **LED**,Vled es el voltaje necesario para el funcionamiento del **LED**, generalmente está entre 1.7 y 3.3 voltios, depende del color del **diodo** y de la composición de metales.

El microcontrolador tiene varios puertos de los cuales podemos hacer uso, estos puertos los podemos configurar como nosotros queramos, como entrada o como salida, para poder hacer esto es necesario escribir en los registros del puerto para darle las instrucciones necesarias.

Existen tres principales formas de controlar los puertos, tomamos como ejemplo el puerto B.

DDR

Para hacer que el puerto B se comporte como entrada o salida, es necesario setear el DDR, este registro no activará ni desactivará ningún pin del microcontrolador, simplemente le indicará al puerto si este será entrada o salida.

The Port B Data Direction Register – DDRB

Bit	7	6	5	4	3	2	1	0	
	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Para indicarle al DDR si el puerto será de entrada o salida, el 1 indica salida, y el 0 entrada, se le puede escribir como hexadecimal, decimal, o binario, por ejemplo, si queremos que todos los bits del puerto sean salidas lo podemos escribir como sigue:

DDRB=0xFF; //Como Hexadecimal, DDRB=255; //Como Decimal, DDRB=0b11111111; //Como Binario

Si queremos que algunos bits funcionen como entradas:

DDRB=0x8C; //Como Hexadecimal, DDRB=140; //Como Decimal, DDRB=0b10001100; //Como Binario

Se puede escribir en cualquiera de los tres tipos (binario hexadecimal y decimal), en todos los registros.

PORT

El PORT controla la salida del puerto, este se usa en caso de que el DDR haya sido seleccionado como salida, un 1 en el PORT indica un nivel alto en el puerto como salida, un 0 indica que el pin estará en nivel bajo. Veamos algunas configuraciones de ejemplo para el PORT:

PORTB=0xFF; Todos los pines están en 1 lógico (high), PORTB=0x00; Todos los pines están en 0 lógico (low)

PORTB=0x03; //Solo los primeros dos bits del puerto están **high**

The Port B Data Register – PORTB

Bit	7	6	5	4	3	2	1	0	
	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PIN

El PIN es un registro de lectura (notar en la imagen del registro donde dice Read/Write, todos son R), este registro nos da un 1 si el pin del microcontrolador tiene una tensión mayor a Vih (limite de tensión de entrada por arriba del cual se considera 1 lógico) , y un cero si el pin presenta una tensión menor a ViL (la tensión de entrada low por debajo de la cual se considera 0 lógico.

En este caso el valor del PIN se le puede asignar a una variable la cual guardará el valor del mismo, al momento de ejecutar la instrucción. ej.

valor=PINB; //El valor de PINB

The Port B Input Pins Address – PINB

Bit	7	6	5	4	3	2	1	0	
	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Aclaración: Cuando el micro resetea, es decir empieza a funcionar, los puertos están seteados como entrada(como valor inicial). DDR valor inicial = 0

Trabajo Práctico N° 2
-Manejo de Puertos

Objetivo:

Usar los registros de los puertos, ver utilidad de R pullup interna

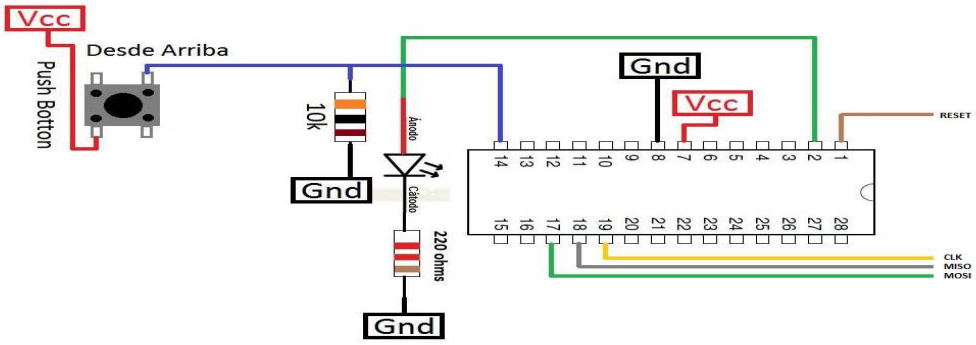
Descripción

1- Hacer un programa que prenda un LED cuando se presiona el pulsador y que lo apague cuando se deje de presionar. El LED está conectado a un pin del microcontrolador y un pulsador a otro pin.
2- Como modifica el circuito y el programa, para usar la resistencia de pullup interna de los ports al conectar el pulsador. Se ahorra algún componente?

Materiales

1 LED
1 Resistencia de 220 Ohms
1 Resistencia de 10Kohms
1 Pulsador
1 Microcontrolador ATmega328p
Programador USBasp V3.0

Diagrama Esquemático



Lectura recomendada

"The AVR microcontroller and embedded systems. Embedded system using Assembly and C". Autores: MUHAMMAD ALI MAZIDI, SARMAD NAIMI, SEPEHR NAIMI
Capítulo 4

Trabajo Práctico N° 3
-AUTO FANTASTICO / Rotación de LEDs

Objetivo:

Avanzar en el manejo de puertos. Poder manejar en forma independiente cada pin. Ver las características DC del micro y analizar cuanta corriente debería entregar el micro, y si es un consumo acorde a las hojas de datos.

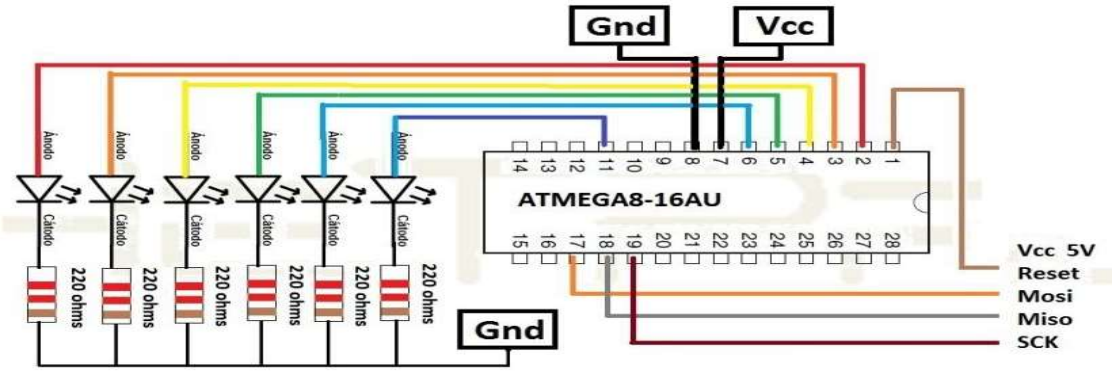
Descripción

Van a conectar al micro 6 leds como indica el esquema. La idea es hacer un programa para prender de a un LED a la vez, e ir desplazando el LED prendido de derecha a izquierda, y viceversa. Solo 1 LED prendido a la vez. Calcular el consumo, corriente por pin y corriente total suministrada por el micro.
Aclaración: los leds deberían prender 100000, 010000, 001000, …….. , 000001, 000010, 000100, etc

Materiales

6 LEDs
6 Resistencias de 220 Ohms
1 Microcontrolador ATmega328p
Programador USB asp V3.0

Diagrama Esquemático:



Lectura recomendada

"The AVR microcontroller and embedded systems. Embedded system using Assembly and C". Autores: MUHAMMAD ALI MAZIDI, SARMAD NAIMI, SEPEHR NAIMI
Capítulo 4

Trabajo Práctico N° 4
-Interrupción Externa

Descripción

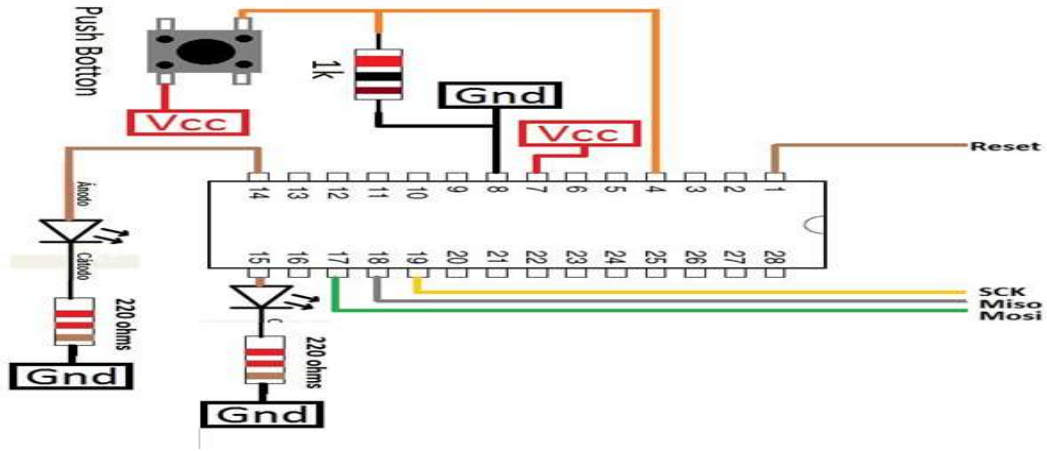
Como indica el esquema, hay un pulsador conectado a la INT0 (INTERRUPCION 0), de forma que al accionar el pulsador se produce una interrupción y 2 LEDs conectados al port B. LED_0 y LED_1.
Se pide hacer un programa que prenda y apague leds siguiendo estas consignas:
- El programa principal va a ser responsable de encender el LED_0
- Se detectará el accionar del pulsador por flanco, la rutina de interrupción debe hacer:
    * apagar el LED_0
    * hacer parpadear al LED_1 5 veces a una frecuencia aproximada de 1Hz
    * prender el LED_0 antes de salir de la interrupción

Resumiendo: cuando resetee el micro penderá el LED_0, cada vez que oprima el pulsador apagará el LED_0 , parpadeará el LED_1 5 veces y prenderá el LED_0.
El circuito armado está configurado con un resistencia de *pull-down*, que modificaría para poner una resistencia de *pull-up*

Materiales

1 Pulsador
2 Resistencia de 220 Ohms, 1 Resistencia de 1 kOhms
2 LED
1 Microcontrolador ATmega328p
Programador USBasp V3.0

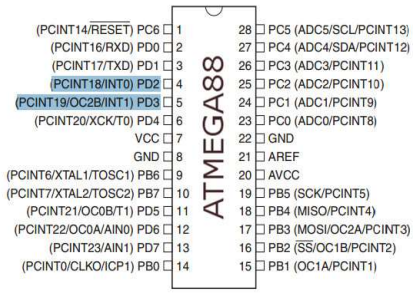
Diagrama Esquemático



Interrupción externa

Las interrupciones externas en el ATmega8 son activadas con los pines INT0 y INT1, en caso de que se habiliten las interrupciones los pines INT siempre activaran alguna interrupción sin importar como se haya configurado el puerto en el que estos pines se encuentren. Las interrupciones externan se habilitan cuando la entrada del pin, cambia de estado, se puede configurar si se requiere que se active cuando cambia de un estado bajo a uno alto o viceversa.

La interrupción externa AVR Atmega 48/ 88/168 /328 se produce cuando en un pin preparado especialmente para este fin se produce algún evento, como por ejemplo ocurra algún cambio de estado en el pin como por ejemplo pasar de un bajo(0) a un alto(1); la interrupción externa AVR se comentará para el caso del ATmega88 pero si se quiere utilizar algún otro micro, el procedimiento es muy similar, solo hay que guiarse de la hoja de datos del microcontrolador AVR utilizado.



La interrupción externa AVR es útil para el manejo de pulsadores, detectores de cruce por 0, teclados matriciales y mucho más; hay 2 tipos de interrupciones externas en los microcontroladores AVR, en la imagen se tiene la representación de los pines (*pinout*) del Atmega88, los pines resaltados que se nombran como INT0 e INT1, son pines que están preparados para producir una interrupción externa AVR por diversos eventos (que se pueden configurar por programa), otros microcontroladores AVR tienen más pines INTx; mientras que los pines nombrados como PCINT0, PCINT1 así hasta PCINT23, son pines que están preparados para producir una interrupción externa AVR, cuando en estos pines se produce un cambio de estado, esto es si sus estados pasan de alto a bajo o de bajo a alto; las interrupción externa AVR se producirá no importando si el pin elegido es una entrada o salida digital.

INTERRUPCIÓN EXTERNA AVR INT0 E INT1

Los pines INT0 e INT1 trabajan de forma independiente pero realizan el mismo tipo de tarea, a través de estos pines se pueden realizar una interrupción externa AVR, siendo la causa o el evento que produzca la interrupción uno de los siguientes motivos:

1. Un 0 o bajo en el pin INT0 o INT1.
2. Por cambio de estado en el pin INT0 o INT1, esto es que pase de un bajo a un alto o de un alto a un bajo.
3. Por flanco de bajada ocurrido en el pin INT0 o INT1, esto es que el estado del pin pase de un alto a un bajo.
4. Por flanco de subida ocurrido en el pin INT0 o INT1, esto es que el estado del pin pase de un bajo a un alto.

Para elegir el tipo evento que producirá la interrupción externa AVR y el pin a utilizar, se utiliza el registro llamado registro de control de la interrupción externa EICRA.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

- Los bits 7 al 4 del registro EICRA no son utilizados les suele poner a 0
- Con los bits 3 y 2 se elige utilizar el pin INT1
- Con los bits 1 y 0 se elige utilizar el pin INT0,

De acuerdo a las combinaciones de bits que se hagan se producirá la interrupción externa AVR por alguno de los eventos mencionados líneas arriba.

En la siguiente imagen se muestran las combinaciones de bits para la elección del evento que producirá una interrupción externa AVR por el pin INT1.

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request
0	1	Any logical change on INT1 generates an interrupt request
1	0	The falling edge of INT1 generates an interrupt request
1	1	The rising edge of INT1 generates an interrupt request

En la siguiente imagen se muestran las combinaciones de bits para la elección del evento que producirá una interrupción externa AVR por el pin INT0.

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request
0	1	Any logical change on INT0 generates an interrupt request
1	0	The falling edge of INT0 generates an interrupt request
1	1	The rising edge of INT0 generates an interrupt request

Para habilitar el uso de la interrupción externa AVR se utilizará el registro llamado EIMSK, además deben estar habilitadas las interrupciones globales, instrucción assembler sei ,lo cual en el ATMEL STUDIO se hará con la instrucción sei(),

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/write	R	R	R	R	R	R	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Los bits 7 a 2 no se utilizan por lo cual se les suele poner a 0, poniendo a 1 su bit 1 se habilita el uso de la interrupción externa AVR del pin INT1; poniendo a 1 su bit 0 se habilita el uso de la interrupción externa AVR del pin INT0.

Para detectar cuando se ha producido la interrupción externa AVR se cuenta con el registro EIFR.

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	–	–	–	–	–	–	INTF1	INTF0	EIFR
Read/write	R	R	R	R	R	R	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

Los bits 7 a 2 no se utilizan por lo cual se les suele poner a 0, para detectar si se ha producido una interrupción externa avr por el pin INT1, el bit 1 se tendrá que poner a 0, cuando se produzca la interrupción este bit se pondrá a 1, si se quiere seguir produciendo mas interrupciones por el pin INT1 este bits habrá que ponerlo nuevamente a 0 dentro de la rutina de interrupciones, cuando se utiliza la rutina de interrupciones con el ATMEL STUDIO, este bits se pone a 0 automáticamente; para detectar si se ha producido una interrupción externa avr por el pin INT0, el bit 0 se tendrá que poner a 0, cuando se produzca la interrupción este bit se pondrá a 1, si se quiere seguir produciendo mas interrupciones por el pin INT0 este bits habrá que ponerlo nuevamente a 0 dentro de la rutina de interrupciones, cuando se utiliza la rutina de interrupciones con el ATMEL STUDIO, este bits se pone a 0 automáticamente;

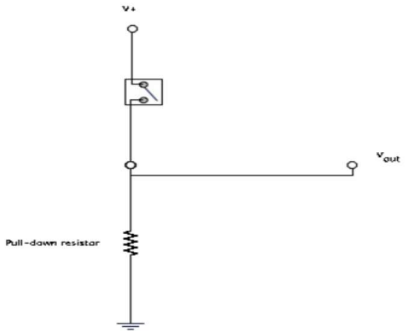
Lectura recomendada

"The AVR microcontroller and embedded systems. Embedded system using Assembly and C". Autores: MUHAMMAD ALI MAZIDI, SARMAD NAIMI, SEPEHR NAIMI

Capítulos 4 y 10

Resistencia de Pulldown

Cuando se conecta un pin del microcontrolador a un *switch*, este al presionarlo nos presenta un nivel alto en el pin. Cuando el *switch* está abierto, la resistencia de *pull-down* nos da un cero o nivel bajo en el pin del microcontrolador.



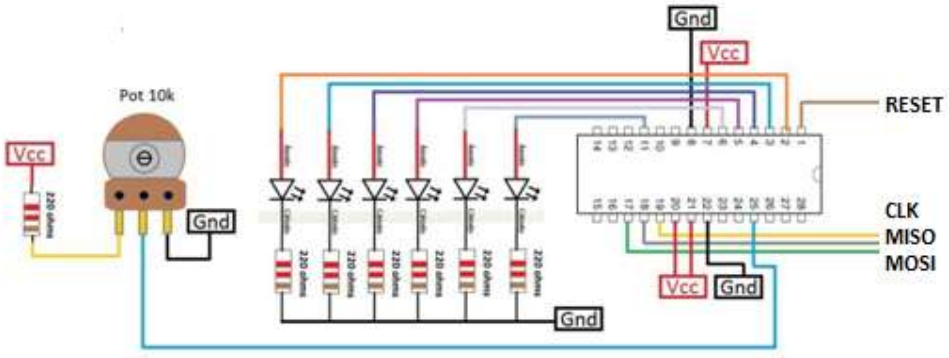
Trabajo Práctico N° 5

- Uso del ADC

Descripción

Con este programa podremos visualizar a través de los LEDs, el valor en binario tomado del ADC conectado a un potenciómetro. El ADC se trabajará a manera de conversión simple y se tomaran solo 8 de los 10 bits, de los cuales se ajustarán para que la salida vaya de 0 a 63, que será representada por los LEDs conectados al microcontrolador.

Diagrama Esquemático



Materiales

1 Potenciómetro de 10K Ohms

6 LEDs

7 Resistencias de 220 Ohms

1 Microcontrolador ATmega8

Programador USBasp V3.0

El ADC

El ADC convierte señales continuas a números discretos. El ADC es un dispositivo electrónico que pasa un nivel de voltaje de entrada a un valor digital proporcional a la magnitud de la entrada, la salida digital puede estar descrita por diferentes codificaciones.

En este caso, el ADC a utilizar es el del microcontrolador ATmega8 el cual es un ADC de 10 bits, de los cuales solo usaremos 8. Las características principales del ADC del ATmega8 son:

-Resolución de 10 bits

-± 2 bits de precisión

-13 a 260 us de tiempo de conversión

-6 Canales de entrada multiplexados

-Rango de voltaje de entrada de 0-Vcc

-Selector de voltaje de referencia de 2.56v

-Tipos de conversión continuo o simple

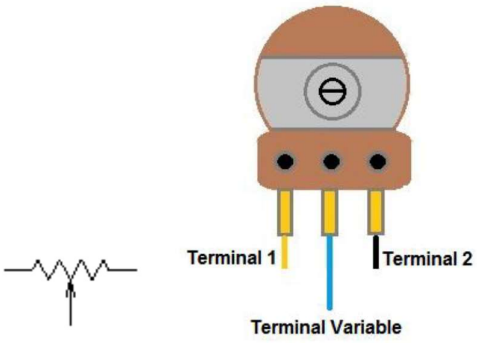
-Interrupción de conversión

-Ahorro de energía

Potenciómetro

El potenciómetro es un tipo de resistencia variable el cual varía conforme se gira la perilla que tiene, en este caso el potenciómetro es usado para generar un divisor de voltaje el cual al variar la resistencia, la salida de voltaje también cambiara proporcionalmente.

El potenciómetro tiene 3 patas las cuales son:



Detalles del programa

ADCSRA = 0xC0;

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Para el registro ADCSRA se asigno 0xC0 o 0b11000000, hexadecimal o binario respectivamente. El bit 7 ADEN habilita el uso del ADC, y el bit 6 ADSC al escribirle un uno inicia la conversión.

ADMUX = 0x22;

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Se activan los bits 5 y 2 por lo que el registro nos queda como 0b00100010 (lo que es igual en hexadecimal a 0x22), al activar el bit 1 le indicamos al ADC que tome la entrada del pin del ADC2

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7

con el bit 5 (ADLAR) del registro ADMUX configuramos la manera en la que nos deposita el valor en los dos registros, para este caso se configuro de la siguiente manera, en la que como se puede ver se ignoraron los dos bits más significativos.

ADLAR = 1

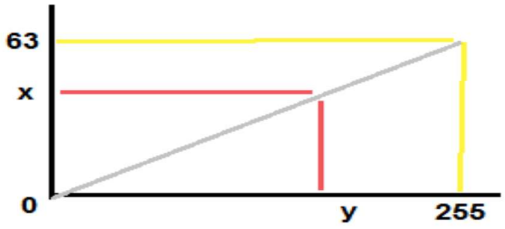
Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

ADCSRA|=_BV(ADSC);

Al estar trabajando el ADC en este modo es necesario indicarle cada cuando tiene que realizar la conversión, con esta instrucción solo entra al bit ADSC del registro y lo habilita, no se modifica cualquier otro valor del registro ADCSRA.

ADC_val=(ADCH*63)/255;

Ya que el valor del ADCH es de 8 bits (como se ve en la imagen al ajustar el ADLAR), se tiene que ajustar la salida a que sea de 6 bits, ya que se están usando solo 6 LEDs, esto se hace ajustando la recta, multiplicando por el máximo de nuestra salida ideal y dividiéndolo por el máximo de la salida obtenida.



Trabajo Práctico N° 6 – Timers

Objetivo

Manejo de registros de timers, generar interrupciones con eventos del timer, manejo de antirrebote de teclas.

Descripción

Se pide hacer un programa que haga parpadear el LED conectado al PB0, en 3 frecuencias distintas o que lo deje ENCENDIDO FIJO, según los valores que haya en las entradas PD0 PD1 según se indica en la siguiente tabla:

PD0	PD1	ESTADO DEL LED
0	0	ENCENDIDO FIJO
0	1	PARADEA CON PRESCALER CLK/64
1	0	PARADEA CON PRESCALER CLK/256
1	1	PARADEA CON PRESCALER CLK/1024

Nota: si alguien presiona las teclas mientras está funcionando el micro, los leds deberán cambiar acorde a la tabla.

Para resolver esta práctica usarán el **Timer1, interrupción por OVERFLOW**
Cuando el LED esté **ENCENDIDO FIJO** el timer estará apagado.
En los otros 3 casos, el timer contará los pulsos de clock divididos por prescaler 64, 256 y 1024 respectivamente.

Cuando se produce un overflow (desborde) deberán cambiar el estado del LED, es decir, si está prendido lo apagan y viceversa

Calculen la frecuencia o periodo con que se prenderá el LED en los 3 casos.

Se les recuerda que si usan la placa de Arduino, la frecuencia del clock es 16MHz.

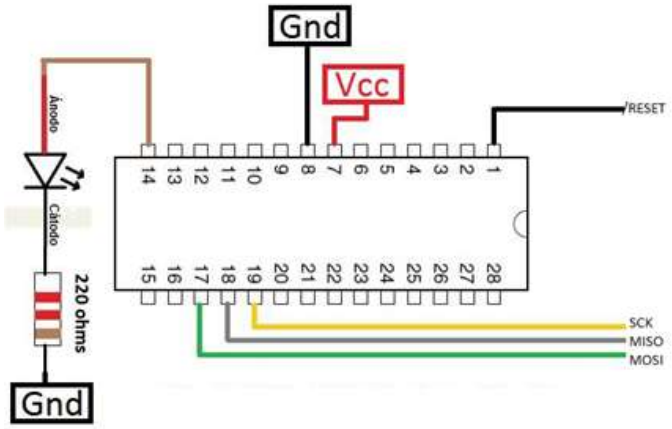
En las entradas PD0 PD1 están conectados 2 switches, que como cualquier tecla produce rebotes al ser presionada. Deben implementar rutinas antirrebote para detectar correctamente las teclas

Materiales

1 LED
2 Switches
2 Resistencias 10 kOhms
1 Resistencia de 220 Ohms
1 Microcontrolador ATmega AVR

Programador USBasp V3.0 (según disponibilidad)

Diagrama Esquemático



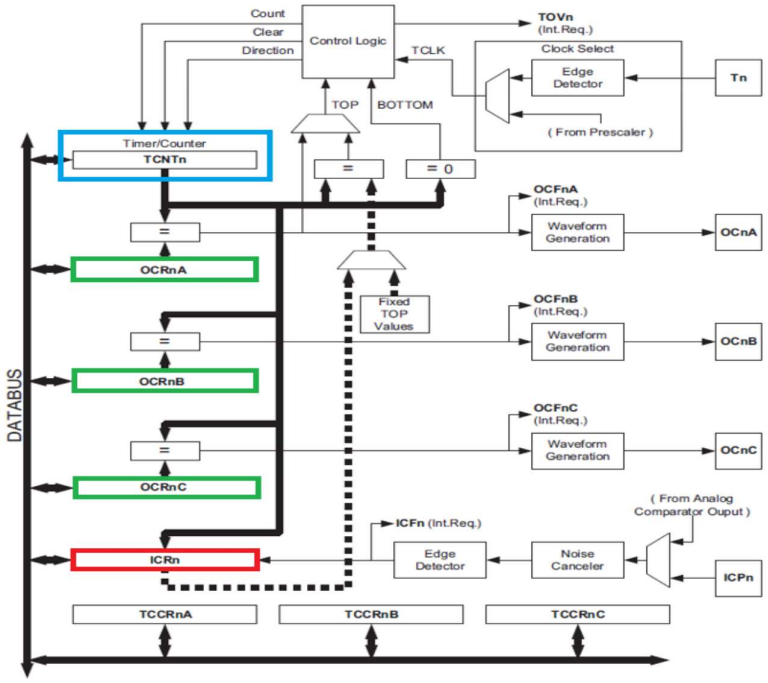
Temporización en AVR

El Atmega328p cuenta con dos timers de 8 bits (TC0 y TC2), con prescaler y modo de Comparación, y un timer de 16 bits (TC1) con prescaler y modos de Comparación y Captura. Estos permiten controlar de manera precisa los tiempos de ejecución de programa (gestión de eventos), así como también generar y medir ondas o señales. Cuenta además con seis canales de PWM, cuatro de 8 bits (asociados a TC0 y TC2) y dos de 16 bits (asociados a TC1).

El Atmega2560 cuenta con dos timers de 8 bits (TCO y TC2), con prescaler y modo de Comparación, y cuatro timers de 16 bits (TC1, TC3, TC4 y TC5) con prescaler y modos de Comparación y Captura. Estos permiten controlar de manera precisa los tiempos de ejecución de programa (gestión de eventos), así como también generar y medir ondas o señales. Cuenta además con cuatro canales de PWM de 8 bits (asociados a TC0 y TC2), y 12 canales de PWM con resolución programable de entre 2 y 16 bits (asociados a TC1, TC3, TC4 y TC5).

7.2.1. Timers de 16 bits

Para los timers TCn (con n=1 en Atmega328p y n=1, 3, 4 y 5 en Atmega2560), es decir, los timers de 16 bits, los registros principales del TCn son TCNTn, OCRnX (con X=A, B en Atmega328p y X=A, B, C en Atmega2560) e ICRn, todos de 16 bits.



- El registro TCNTn contiene el valor del timer en cada instante (se puede escribir y leer).

- Los registros OCRnX (Output Compare) se escriben con un valor que será comparado permanentemente con el valor de TCNTn. Los comparadores dan a su salida un '1' sólo en caso de coincidencia o "match" entre sus entradas, y dan un '0' para el resto del tiempo. La coincidencia levanta además un flag (OCnX) que puede usarse para interrupción.

- El registro ICRn (Input Capture) "congela" el valor de TCNTn en el momento en el cual se produce el evento configurado en el "Edge detector" (con el bit ICESn del registro TCCRnB) en el pin ICPn. A su vez, en ese momento se levanta un flag de captura ICFn, permitiendo ir a su rutina de servicio (si la correspondiente interrupción está habilitada) a leer el valor "congelado" en ICRn.

Existen distintos modos de operación para el TCn, los cuales se eligen escribiendo los bits WGM10:3 de los registros TCCRnX.

Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	-	-	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Table 20-7. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0		1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	clkI/O/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)

Uso de los bits **COMnX**[0:1]:

Table 20-3. Compare Output Mode, non-PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on Compare Match.
1	0	Clear OC1A/OC1B on Compare Match (Set output to low level).
1	1	Set OC1A/OC1B on Compare Match (Set output to high level).

Table 20-4. Compare Output Mode, Fast PWM

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM1[3:0] = 14 or 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at BOTTOM (non-inverting mode)
1	1	Set OC1A/OC1B on Compare Match, clear OC1A/OC1B at BOTTOM (inverting mode)

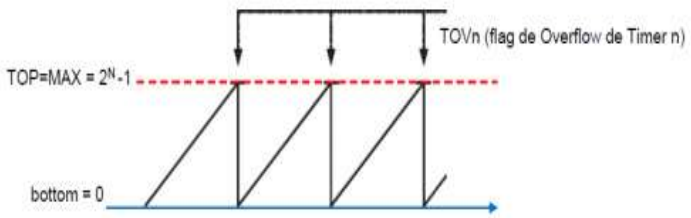
Table 20-5. Compare Output Mode, Phase Correct and Phase and Frequency Correct PWM

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM1[3:0] = 9 or 11: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match when up-counting. Set OC1A/OC1B on Compare Match when down-counting.
1	1	Set OC1A/OC1B on Compare Match when up-counting. Clear OC1A/OC1B on Compare Match when down-counting.

Algunos de los modos de temporización para los timers de 16 bits son los siguientes:

MODO NORMAL (0)

El timer cuenta desde $\mathbf{0}$ hasta su valor máximo ($\mathbf{2^{16} - 1}$), desborda y vuelve a $\mathbf{0}$, levantando el flag de desborde TOVn. Es decir, desborda cada $\mathbf{2^{16}}$escalones del timer, cada uno de los cuales se mantiene durante un tiempo Tcy = $1/FCPU$ e inmediatamente se incrementa.
Si se habilita la interrupción por desborde, podemos cambiar el estado de un bit en la rutina de servicio para verificar el tiempo transcurrido. Por ejemplo, para una frecuencia de reloj de 16MHz, con prescaler en 1, el tiempo entre desbordes será de:



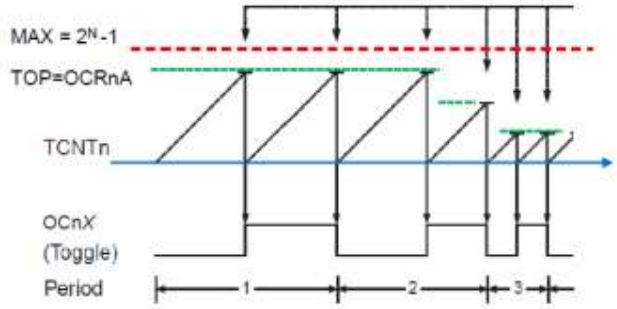
$$t = \frac{(2^{16})\ ciclos}{(16 * 10^{6})\ ciclos/seg} = 4.096ms$$

Con prescaler en 8, el tiempo será de $\mathbf{8}*\mathbf{4}.\ \mathbf{096}\boldsymbol{ms}$ = $\mathbf{0}.\ \mathbf{0328}\boldsymbol{s}$, etc

MODO CLEAR TIMER ON COMPARE MATCH o CTC (4)

El timer cuenta desde 0 hasta el valor TOP del registro OCRnX, se resetea y levanta el flag de coincidencia OCnX. Puede actuar también sobre el pin OCnX. Para lograr un período de N pulsos (N de 2 a 65536) debe hacerse

OCRnX = N-1.



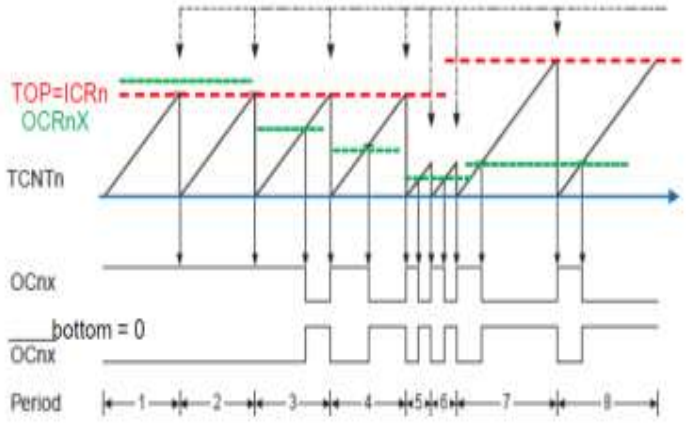
Si se habilita la interrupción por coincidencia, al igual que en el caso anterior, podemos cambiar el estado de un bit en la rutina de servicio para verificar el tiempo transcurrido. Pero una alternativa más fácil es activar el cambio de estado de OCnX en la configuración de los registros. Por ejemplo, para una frecuencia de reloj de 16MHz, con prescaler en 1, si se quiere lograr un tiempo entre coincidencias de 2.5ms, entonces:

$$\frac{(OCRnX + 1)\ ciclos}{(16 * 10^6)\ ciclos/seg} = 2.5ms$$

$$OCRnX = 39999$$

Con esta configuración y prescaler en 256, el tiempo será de $256 * 2.5ms = 0.64s$, etc.

MODO FAST PWM



El timer cuenta desde 0 hasta el valor TOP del registro ICRn y se resetea. Además, si la salida OCnX está configurada como “clear”, ésta se pone en ‘1’ al comienzo de cada período y se pone en ‘0’ en cada coincidencia del timer con ICRn. Si la misma se configura como “set”, actúa en forma opuesta. En definitiva, permite fijar el período con ICRn y el duty-cicle (tiempo en alto) con OCRnX.

Por ejemplo, para una frecuencia de reloj de 16MHz, con prescaler en 1, si se quiere lograr una señal de PWM al derecho, con un período fijo de 2.5ms, entonces:

$$\frac{(ICR1 + 1)\ ciclos}{(16 * 10^6)\ ciclos/seg} = 2.5ms$$

$$ICRn = 39999$$

Además, la salida de OCnX deberá estar configurada como “clear”.

MODO CORRECT PHASE PWM

El timer cuenta desde 0 hasta el valor TOP del registro ICRn y vuelve en forma descendente hasta 0 nuevamente. Además, la salida OCnX conmuta en cada coincidencia del timer con ICRn. Permite fijar el período con ICRn y el duty-cicle (tiempo en alto) con OCRnX.

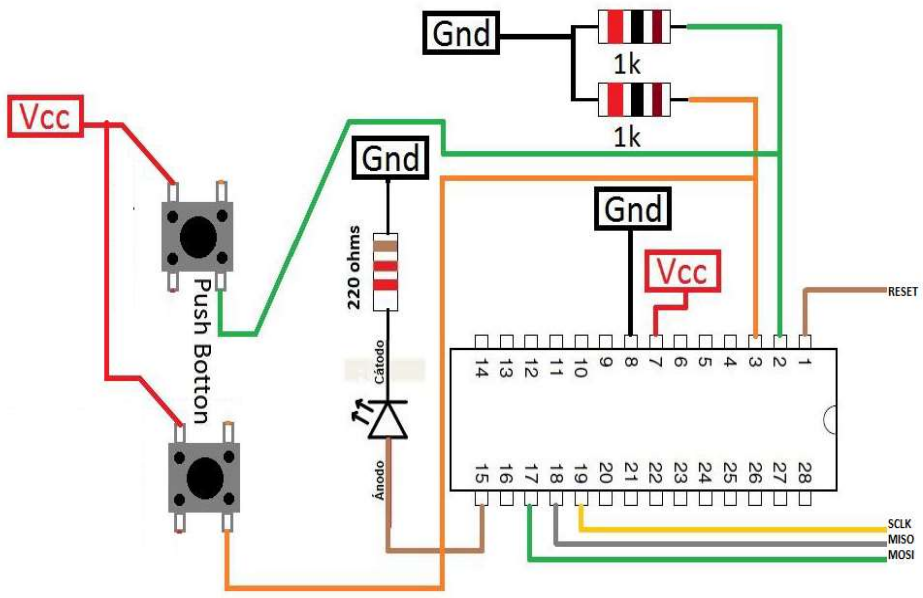
Trabajo Práctico N° 7 – PWM

Descripción

Hacer un programa que aumente y disminuya el brillo de un LED. Para eso se dispone de 2 pulsadores (UP, DOWN) y un LED como indica el esquema.

Deberán usar el PWM, o modulación por ancho de pulso, la cual consiste en modificar el ciclo de trabajo de una señal (sin modificar la frecuencia). Con esta señal se alimentará el LED, de forma que el valor medio de la señal será proporcional al brillo del LED, a mayor ancho de pulso, más brillo

Diagrama Esquemático

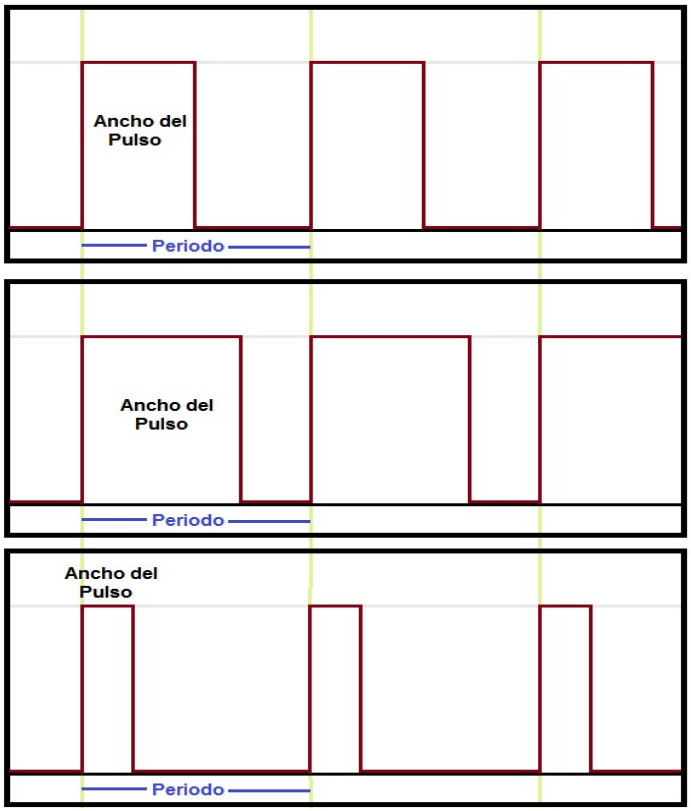


Materiales

1 LED
1 Resistencia de 220 Ohms
2 Pulsadores
2 Resistencias de 10Kohms
1 Microcontrolador ATmega8
Programador USBasp V3.0

PWM

Modulación por ancho de pulso o PWM (Pulse Width Modulation), de una señal, es cuando se modifica el ciclo de trabajo o el ancho del pulso de una señal periódica, en este caso representado por una señal cuadrada, uno de los usos del PWM entre muchos otros, es controlar la cantidad de energía, en este caso el voltaje promedio es mayor conforme aumenta el ciclo de trabajo.



En la imagen se puede observar, que el período de la señal permanece fijo, por lo tanto, la frecuencia también, solamente cambia el ciclo de trabajo, en la primera se observa que el ciclo de trabajo es de aproximadamente 50% lo cual nos indica que es el porcentaje de voltaje promedio entregado a la carga.
El PWM se puede utilizar en varias cosas, como el control de la velocidad de motores de DC, la posición de un servomotor, fuentes conmutadas, entre otras cosas más.

Lectura recomendada

" The AVR microcontroller and embedded systems. Embedded system using Assembly and C". Autores: MUHAMMAD ALI MAZIDI, SARMAD NAIMI, SEPEHR NAIMI

Capítulos 4 y 16

Trabajo Práctico N° 8 – Puerto serie

Descripción

El objetivo de este práctico será establecer comunicación bidireccional serie entre un microcontrolador AVR de 8 bits y una computadora de escritorio.

1) Al encender el microcontrolador, el programa deberá transmitir el texto

**** Hola Labo de Micro ****

Escriba 1, 2, 3 o 4 para controlar los LEDs

El texto de arriba deberá mostrarse en el terminal serie

2) Si en el terminal serie se aprieta la tecla '1', entonces se enciende/apaga el LED 1 (toggle). Si se aprieta la tecla '2', ocurre lo propio con el LED 2 y así lo propio para los cuatro LEDs.

Sugerencias:

- revisar la tabla ASCII
- considerar el agregado de un *delay* en el inicio para poder abrir el terminal serie

Lectura requerida

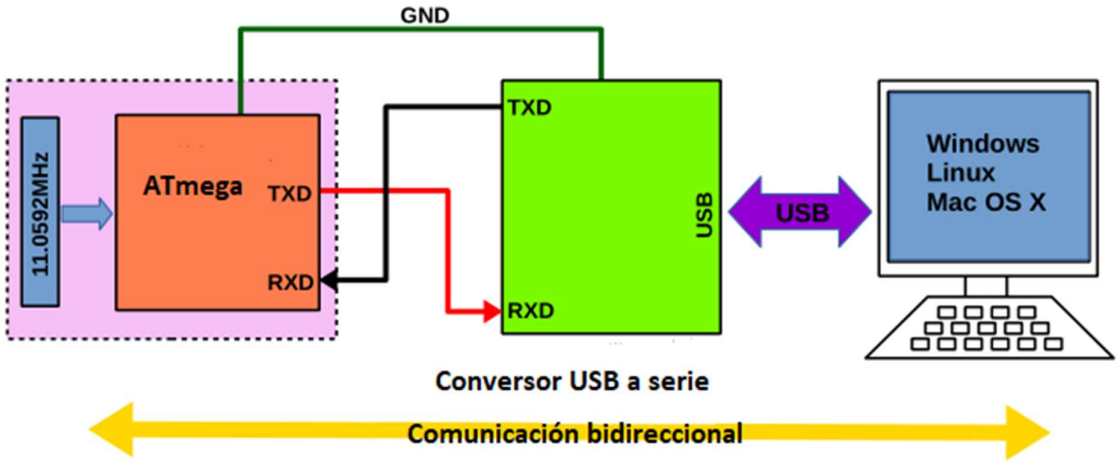
"The AVR microcontroller and embedded systems. Embedded system using Assembly and C". Autores: MUHAMMAD ALI MAZIDI, SARMAD NAIMI, SEPEHR NAIMI
Capítulo 11

Materiales

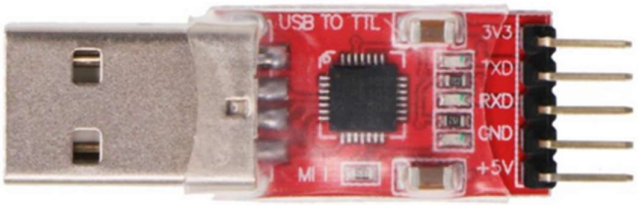
4 LEDs

4 Resistencias de 220 Ohms
1 Microcontrolador ATmega8 PDIP
Programador USBasp V3.0
Conversor TTL a USB (leer el comentario en referencia a las placas Arduino antes de comprar).

La mayoría de las computadoras modernas carecen de puerto serie, en tanto ofrecen conexión USB (Universal Serial Bus). Si bien no es posible conectar directamente un puerto serie a un puerto USB, sí es posible intercalar un hardware que realiza la conversión en ambos sentidos, como indica la ilustración.



El hardware se llama Conversor USB a serie, y en este caso nos conviene obtener uno que tenga salida TTL, no RS232.



En el caso de utilizar una placa Arduino Uno o Arduino Nano (microcontrolador ATmega328p) la funcionalidad de convertir a USB, ya está disponible, dado que se utiliza el mismo hardware para el puerto serie y para reprogramar la placa. En dicho caso, no se necesita comprar el conversor TTL a USB. Solo debe adecuar el circuito de arriba para que funcione con dicho microcontrolador.

Marco conceptual y desarrollo

El puerto serie es un periférico que necesita configuración antes de utilizarse. La configuración necesaria tiene que ver con acordar entre ambos extremos de qué modo se van a serializar los datos. Es decir, como

mínimo:

- A qué velocidad transmitir
- De qué modo sincronizamos lo transmitido con lo recibido
- De qué tamaño es la unidad básica de información comunicable, o “caracter”

Para este Trabajo Práctico, adoptaremos una velocidad de 9600 bits por segundo o *baudrate*. Esto significa que al dar la orden de transmitir o recibir un caracter, sabemos que los dígitos binarios se van a transmitir a razón de 1/9600 segundos.

Respecto a la sincronización, adoptaremos el mecanismo más común, la comunicación asincrónica. Esto significa que no se tendrá una señal de clock que acompañe a los datos, sino una estructura de datos adicional que enmarca la información y permite sincronizar transmisor con receptor. El enmarcado consiste en que, para cada caracter se antepone un bit de inicio (*start bit*), que siempre vale “0” y luego de cada caracter se adiciona un bit de fin (*stop bit*) que siempre vale “1” y coincide con el estado en el que queda el canal cuando no se transmite nada.

En lo que respecta al largo del caracter, adoptaremos 8 bits de datos. En la literatura esto se encuentra como 8N1: 8 bits de datos, sin paridad y 1 bit de stop. El concepto de paridad tiene que ver con agregar un noveno bit a modo de redundancia y así detectar algunos errores del lado de receptor. En este Trabajo Práctico, como mencionamos, no utilizaremos bit de paridad.

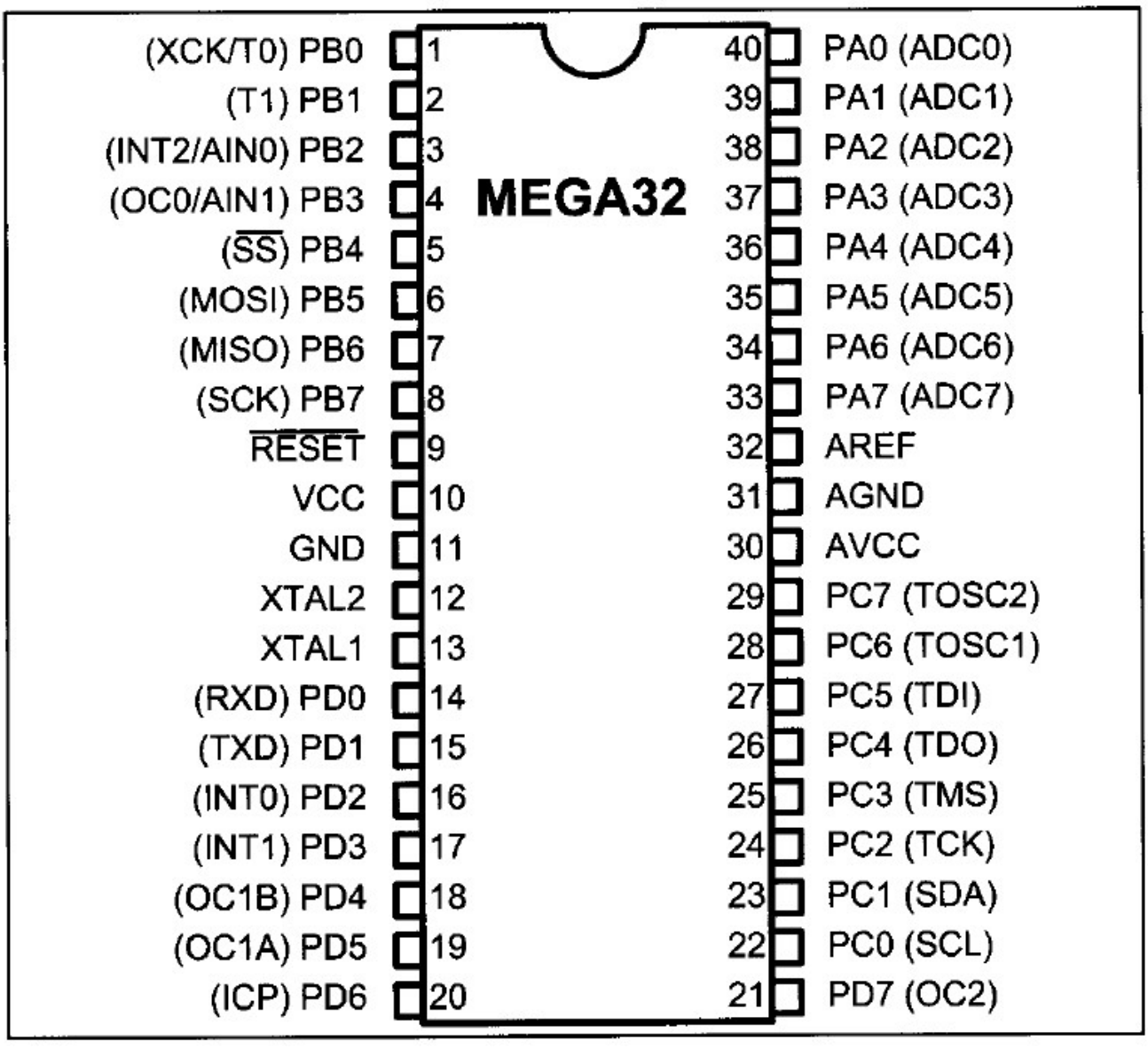
Si sumamos el bit de *start*, los 8 bits de datos y el bit de *stop*, entonces, cada byte de información útil se convierte en 10 bits comunicados. Entonces, el tiempo de transmisión del byte será 10/9600 segundos.

La configuración de registros requeridos para puerto serie está claramente explicada en la bibliografía mencionada.

Como actividad adicional, una vez realizado el programa requerido en el enunciado, proponemos implementar la misma funcionalidad pero utilizando la técnica de interrupciones, en este caso en relación a la recepción de los datos por puerto serie. Es decir: en vez de consultar permanentemente si hay un dato nuevo en el puerto serie, la recepción de un byte deberá generar una interrupción de puerto serie, que al atenderse procesará el dato de la forma correspondiente. Compare ambos loops del programa principal (con y sin interrupciones).

Apéndice

Pinout ATmega32



Pinout ATmega328/p

Pinout

Figure 5-1. 28-pin PDIP

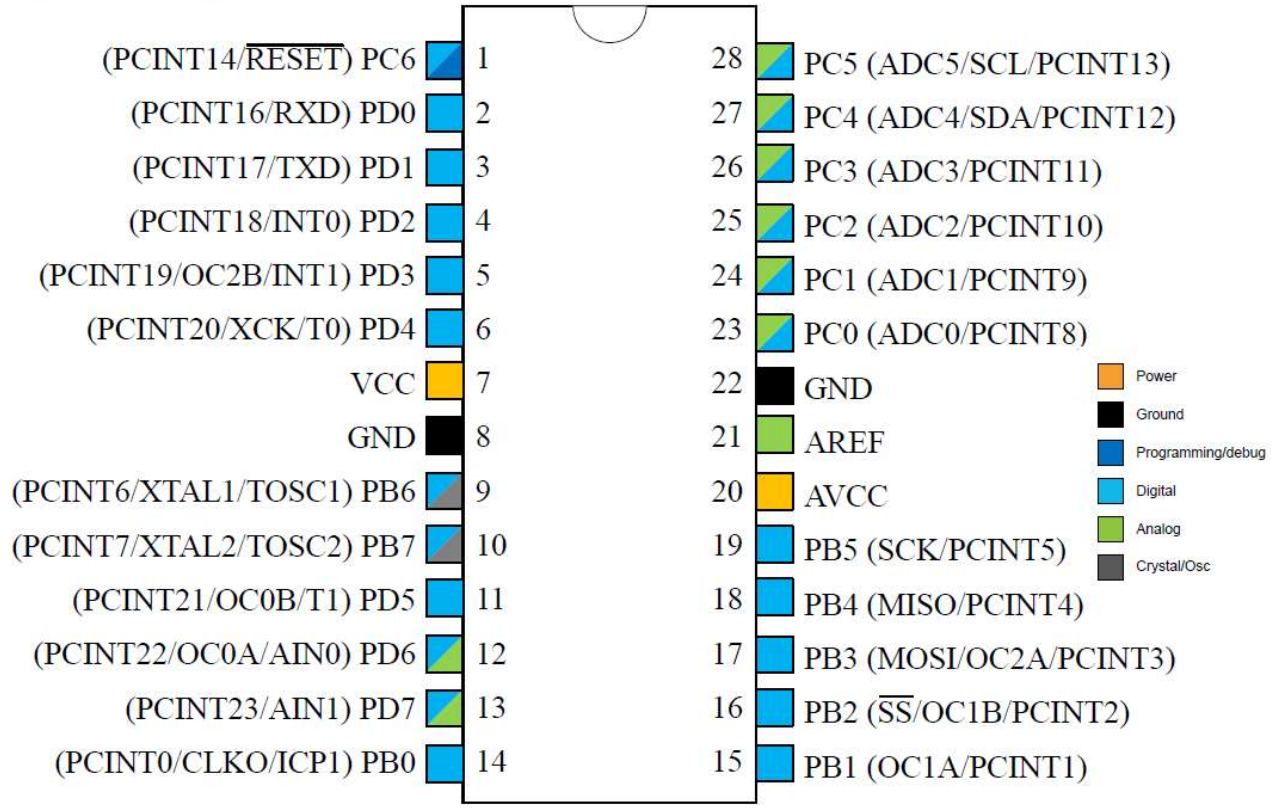


Diagrama de Arduino Uno