Laboratorio 2:

Actividad 1:

1) AnalogRead:

Lee un valor de un pin analógico especificado. La placa de arduino contiene un conversor de analógico a digital de 10 bits de 6 canales (8 canales en el mini y el nano, 16 en el mega). Esto significa que mapeará voltajes entre 0 y 5 volts a valores enteros entre 0 y 1023. Esto lleva a una resolución entre las lecturas: 5volts/ 1024 unidades, o 0.0049 volts (4,9Mv) por unidad. El rango de input y resolución se los puede cambiar con analogReference()

Toma alrededor de 100 microsegundos (0.0001s) para leer una entrada analógica, entonces el máximo rate de lectura es de 10.000 veces por segundo.

LCD:

El pin rs se utiliza para indicar si es señal (comunicarse y hacer algo con el micro, por ej, cortar todo) o si es dato.

Mide un valor analógico de tensión. Tiene un problema, que pasa si apretas dos, te da cualquier cosa.

Digamos, cada pulsador tiene un valor de resistencia asignado, ese valor de resistencia como es un divisor resistivo genera un valor distinto de tensión, después esos valores de tensión son capturados por el conversor, y a partir de dicho valor de tensión te dice su respectivo valor digital.

Tiene la ventaja de que tengo varios pulsadores en un solo puerto, el A0, a diferencia del botón pulsador en el que tenía uno en un puerto en vez de varios.

Tenes una desventaja, necesita un dispositivo para sincronizar los valores de tensión, necesita el conversor analógico digital para poder pasar a analógico, y tiene un tiempo de conversión, y si queres usarlo para otra cosa, necesitas multiplexar también.

Por ejemplo, si tuviera un sensor de temperatura, y el display, (habiendo agregado otro dispositivo), necesito un multiplexor para poder controlar las señales de cada uno de los dispositivos. Además que necesito sincronizar en la programación para saber que estoy midiendo en cada vez que se obtiene un valor del ADC

3) analogWrite: escribe un valor analógico (Onda PWM) a un pin. Puede ser usada para prender un LED a diferentes brillos o variar un motor a varias velocidades. Luego de llamar a analogWrite(), el pin va a generar una señal cuadrada fija continua, con un duty cicle especificado hasta la próxima llamada a analogWrite( o digitalRead o digitalWrite en el mismo pin).

La frecuencia de la señal PWM en la mayoría de los pins es 490Hz. En el uno y otras placas similares los pines 5 y 6 tienen una frecuencia de aprox 980Hz.

En la mayoría de las placas arduino, esta función corre en los pines 3, 5, 6, 9, 10 y 11.

The analogWrite function has nothing to do with the analog pins or the analogRead function.

Manera utilizada en la función analog write para controlar la intensidad del brillo:

El analog write genera una onda cuadrada.

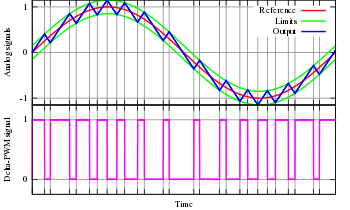
PWM es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (una [senoidal](https://es.wikipedia.org/wiki/Sinusoide" \o "Sinusoide) o una [cuadrada](https://es.wikipedia.org/wiki/Onda_cuadrada), por ejemplo), ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

El duty cycle o el ciclo de trabajo, ciclo útil o régimen de trabajo es la relación que existe entre el tiempo en que la señal se encuentra en estado activo y el periodo de la misma. El ciclo útil D es definido como la relación entre la duración del pulso τ y el período (T) de una onda cuadrada.

En la función analog write, con variar el parámetro value, lo que se varía es el pulse width modulation, que hace que aumente el duty cicle, el largo del pulso en relación con el período (osea tengo un 1 en voltaje que dura más tiempo), como es en continua y tengo niveles de voltaje, y al estar a una frecuencia muy alta, entonces sucede que se mantiene siempre prendido, digamos, el valor lógico en que se termina manteniendo casi siempre es en 1, osea tiene un pulso casi constante

Si yo le bajo el valor, el pwm (el duty cycle de este) baja y se hace menos intensa.

Para generar una señal analógica mediante un pin de salida digital, es necesario utilizar un conversor DAC, que convierta de analógica a digital



Actividad 2:

Para el pin A2 veo que le corresponde el PC2 del atmega, que se corresponde con el PCINT10.

Para este ejercicio, utilizamos las interrupciones externas (que solamente trabajan sobre 2 pines, los INT0 e INT1) y las pin change interruptor (que trabajan sobre un grupo de pines, sobre los que puedo atender bytes, y tienen 23 pines, los PCINTx (0 a 7, 8 a 14 y 16 a 23), donde se dividen en 3 grupos, a su vez cada grupo le corresponde una interrupción, por ej al primer grupo le corresponde la interrupción PCINT0, al segundo PCINT1, y el tercero PCINT2.

Cuando nos piden que configuremos las pin change, tenemos que elegir los registros, habilitarlos con la mascara, elegir ahí que pines van a actuar según poner en 1 los bits de los registros, sobre que pines queremos que sucedan las interrupciones y la rutina ISR (el código dentro de esta y definir que ISR es, si es la pcint1 o pcint0, al grupo que corresponda)

Otra cosa importante para rescatar de acá es que podemos en el código modificar algunos bits para parar el timer (son los bits marcados en el código) y para poder continuar, mediante la división con el prescaler por 1024 y así setear el ciclo que necesita el timer cumplir, luego hace overflow y genera la interrupción y se resetea.

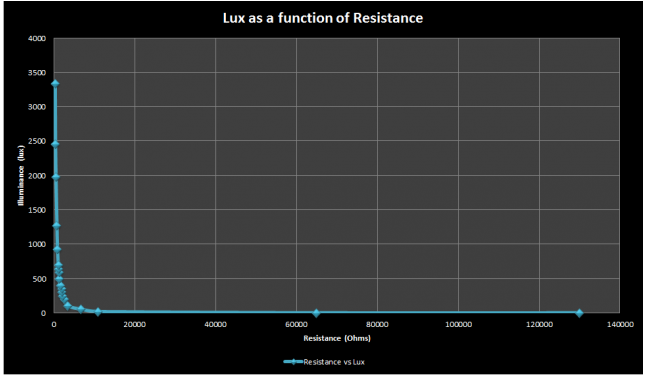
En este ejercicio además utilizamos el timer para que cuando llegue a cierto valor tire la interrupción, lo configuramos para que cada 0,01 segundos tire la interrupción, que timer utilizar? El que corresponda, hay que usar el timer 0 porque con usar el timer 1, necesitamos utilizar la función analogWrite para variar el brillo, y a su vez la estoy usando para el cronómetro, entonces uso los dos para ambos, tendría que cambiar el timer que uso como cronómetro.

Actividad 3

2) Un luxmeter es un dispositivo que mide la iluminación y la emición de luz utilizando una unidad llamada Lux. Efectivamente mide la cantidad de energía de una luz cayendo en una unidad de área. Mide cuan brillosa es la luz cayendo en un sensor.

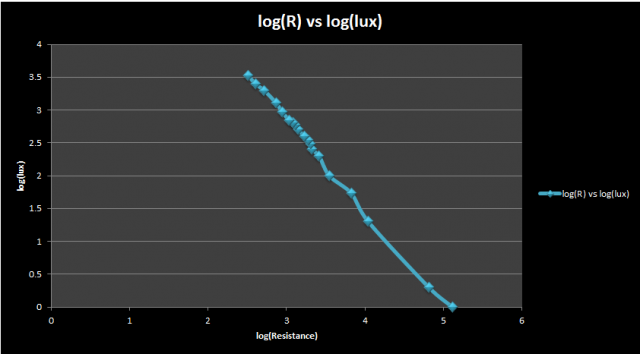
El sensor de luz para el medidor luz, el más fácil de usar y más disponible es el fotoresistor o resistor dependiente de la luz LDR. La resistencia de un LDR cambia con la cantidad de luz que cae en él. Midiendo la resistencia y sabiendo las características particulares del LDR se puede medir la cantidad de luz cayendo en el LDR. Cuanto más brillosa la luz, menor la resistencia. Esta relación no es lineal, sino exponencial.

Se obtuvieron las siguientes medidas del LDR:



This graph indicates that resistance decreases exponentially as the brightness of the light increases.

It turns out that if you take the logarithm of both variables (illuminance and resistance) and then re-plot, you will end up with a (more or less) straight line.  The base of the logarithm could in theory be anything, but I will use a base-10 log.  Here is the resulting plot of log of the lux as a function of the log of the resistance.



Cuanto mayor luz le pegue al LDR, menor será la resistencia.

Cuando menor es la luz, mayor es el valor de la resistencia.

La función obtenida es:

Reemplazando con valores obtengo:

m=-1,437

b=6,727

Y desarrollando esta ecuación obtengo:

Donde el valor de R lo hago variar con el dispositivo LDR.

3) El canal 1 es el A1 del arduino.

El segundo paso es el conversor ADC, tiene un valor de sensor que obtiene a través de analogRead(pin A1) desde el pin A1.

Luego, con lo obtenido anterior, llamamos a obtenerResistencia(), una vez obtenido esto, llamamos a obtenerLux() que es el que mostramos en el LCD.

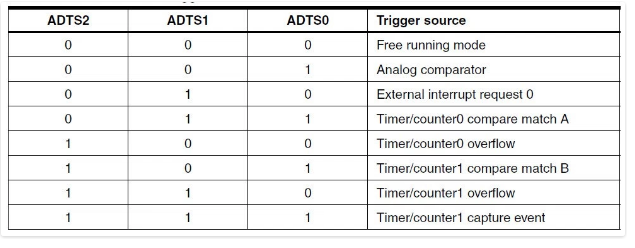
4) Hay que hacer lo mismo que en el inciso 3 pero sin el analogRead()

El adc interno tiene una resolución de 10 bits, eso significa que la tensión analógica de entrada se convierte en un valor entre 0 y 1023.

La entrada en este caso es siempre en el puerto C.

La tensión de referencia la podemos modificar, con el ADMUX.

Hay dos modos, single conversión y free running mode.



Con single conversión el dispositivo se apropia del conversor y si otros dispositivos quieren usar el conversor. Con free running mode, va tomando los valores para convertir de manera continua

**ADCMUX:**

Este registro nos permite especificar qué referencia utilizar, si queremos los resultados justificados a la izquierda, y qué canal del ADC queremos convertir.

MUX3 a MUX0 determina qué canal utilizar. Utilizaremos 0 a 7 para los canales 0 a 7, utilizaremos 8 para el sensor de temperatura interno, o 14 para medir la tensión de referencia.

Nota: si medimos canal 14 usando como referencia Vcc, podremos medir la tensión de alimentación.

Hay que crear una estructura adc\_conf que va a tener el canal y una función de callback.

Struct adc\_conf{

Int canal;

Void (\*callback)();

}

Importante aquí son los paréntesis al puntero, indicando que es función. Luego tiene las siguientes funciones:

adc\_init( adc\_conf \* cfg)

adc\_loop( )

Existirán un adc.h, un adc.c y un main.c, donde en el adc.h estará la estructura y las funciones con las que voy a llamar a otros programas (adc\_init, y adc\_loop).

En el adc.c desarrollamos el adc\_init, y este adc\_init inicializa el conversor analógico digital (tocar registros, como ADMUX, ADCSRA, ADCSRB ADCH, ADCL), en estos registros ADCH y ADCL es donde el conversor va a dejar el valor analógico en binario, y con una resolución en 10 bits.

Y el adc.c también implementa la función adc\_loop que es donde llamamos a la función de callback(), esa función es un procedimiento para obtener el valor de Lux que nosotros estamos midiendo.

La función de callback esta implementada en el programa principal

En el programa principal estará, el main, y dentro del main, llamamos a adc\_init() y un while, donde pondré adc\_loop continuamente.

El conversor analógico digital, lee desde la entrada A1, por aproximaciones sucesivas, este conversor transforma este valor en un valor, hay un tiempo que demora en convertir el valor, cuando se consigue el valor llamamos a la rutina que obtiene dicho valor, que se llama ISR(adc\_vect), que es llamada cuando el conversor termino de convertir el valor, se obtiene el valor en binario, esa rutina se va a ejecutar.

La isr(adc\_vect)

Acá además utilizamos un puntero a funciones,

6) no te conviene llamar a otra función desde la interrupción porque no sabes que puede tener dicha función, puede tener una interrupción adentro y se cuelga, la idea es que no se sabe lo que hay dentro de la función esta. Con la call back sucede lo mismo,

Actividad 4:

La manera mejor de configurar el ADC es single conversión mode, que digamos, procesa un valor, lo convierte, y se detiene, porque si lo hace continuamente un solo dispositivo se adueña del conversor, y no da espacio a otro, por lo tanto, dicho dispositivo se queda con el conversor

Esto funcionaria así:

Elijo dispositivo por interrupción o prioridad como sea,

Modificamos los bits de CSRB para ponerlo en modo single conversión.

Luego con los otros registros indico que arranque.

En el admux tendre que poner varios bits correspondientes a los canales en 1 para poder sensar

Tendré varías rutinas de interrupciones, una por cada dispositivo