dd

Sistemas Operativos Avanzados

Electrónica y Arduino en TinkerCad.

Ing. Waldo A. Valiente

Electrónica y Arduino en TinkerCad

1. Contenido

2. Introducción

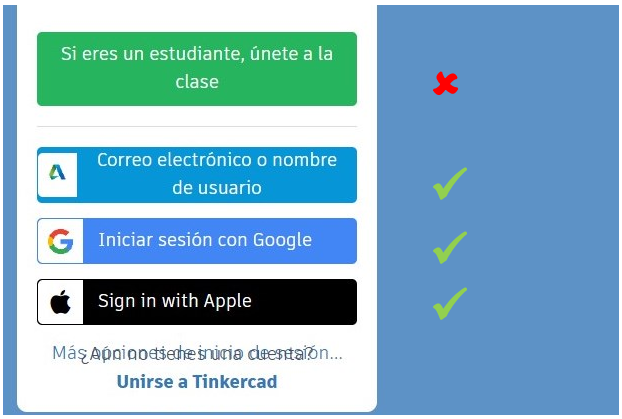
Este apunte busca reunir conceptos teóricos y prácticos sobre electrónica y Arduino. Cuya finalidad es que los alumnos tengan una base académica, en la cual desarrollar sus prácticas sobre los sistemas embebidos que no poseen sistema operativo.

3. TinkerCad

TinkerCad (https://www.tinkercad.com/) es un software gratuito online creado por la empresa AutoDesk[1], una de las empresas dominantes en el software de diseño 3D. La elección de la cátedra en utilizar esta herramienta radica en la sencillez de su uso. Que incluye la posibilidad de simular un prototipo tanto desde el punto de vista eléctrico, lógico y de de funcionamiento. Además dispone de una amplia variedad de componentes sensores y actuadores, que permiten la creación de sistemas embebidos que están a la altura de las necesidades de la cátedra.

La herramienta TinkerCad posee varias funciones, que van desde diseño en tres dimensiones de “cualquier cosa”, tanto sea para componentes a imprimir en 3 dimensiones, pasando por la construcción de bloques tipo Rasti™, a la creación de modelos para Minecraft. También permite codificar desde bloques de código (sin teclear). Nosotros les pediremos que usen la parte de circuitos electrónico “circuits” que permite ensamblar, programar y simular circuitos.

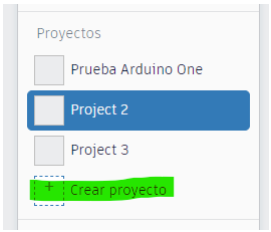
Al ingresar van a tener que crear una cuenta o pueden utilizar la forma de validación de acceso a través de cuentas externas como Google, Apple o cuenta de AutoDesk. No utilicen la opción “Si eres un estudiante, únete a la clase”, ya que la plataforma da la posibilidad de armar aulas virtuales. Ya que en la cátedra usamos *Teams*, por disposición de las autoridades.



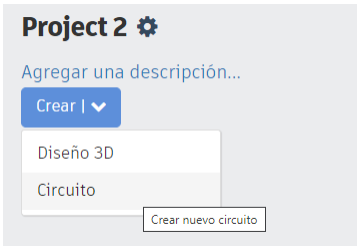
3.1. Nuevo proyecto de circuito

En el menú de la izquierda deben ir a la parte de [Proyecto] y abajo se encuentra el botón [Crear proyecto]. Por defecto lo creará con el nombre “proyecto X”:

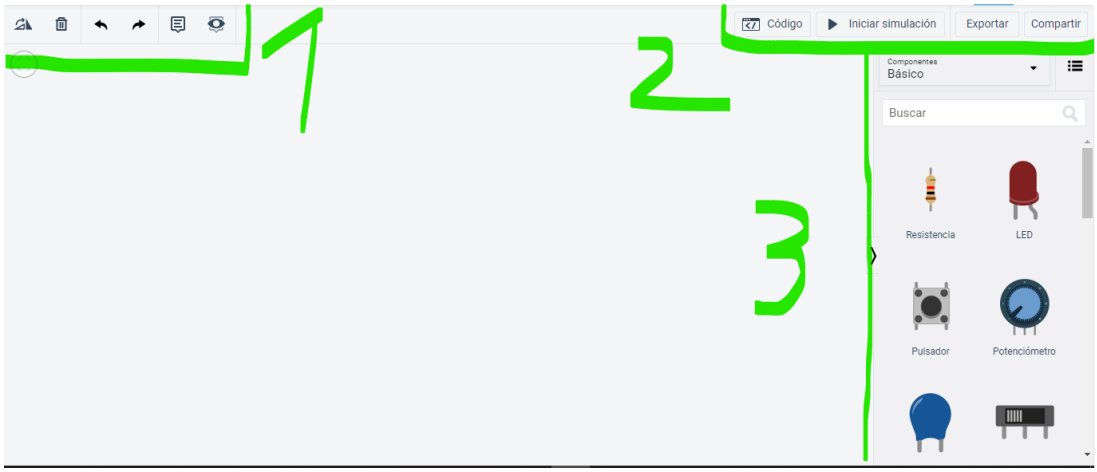
[1] http://www.autodesk.es/



parándose sobre el proyecto, el nombre se puede editar desde el icono de opciones (con forma de engranaje). Abajo se encuentra el Combo desplegable [Crear | v] eligen la opción [Circuito]



En la nueva ventana de trabajo que se despliega se divide en tres partes y tiene la siguiente forma:



La primera sección, está destinada a la edición de los componentes, los botones permiten (de izquierda a derecha) girar, eliminar, deshacer el último cambio, rehacer el último deshacer, insertar anotaciones sobre el tablero, ver u ocultar las anotaciones.

La segunda sección, está destinada al manejo del circuito, estas son (de izquierda a derecha) codificar comportamiento (Se requiere que se encuentre en el tablero un componente eléctrico tipo microcontrolador), iniciar la simulación del circuito electrónico, exportar el circuito en un archivo con extensión .BRD, compartir permite generar/invitar personas para ver el circuito creado.

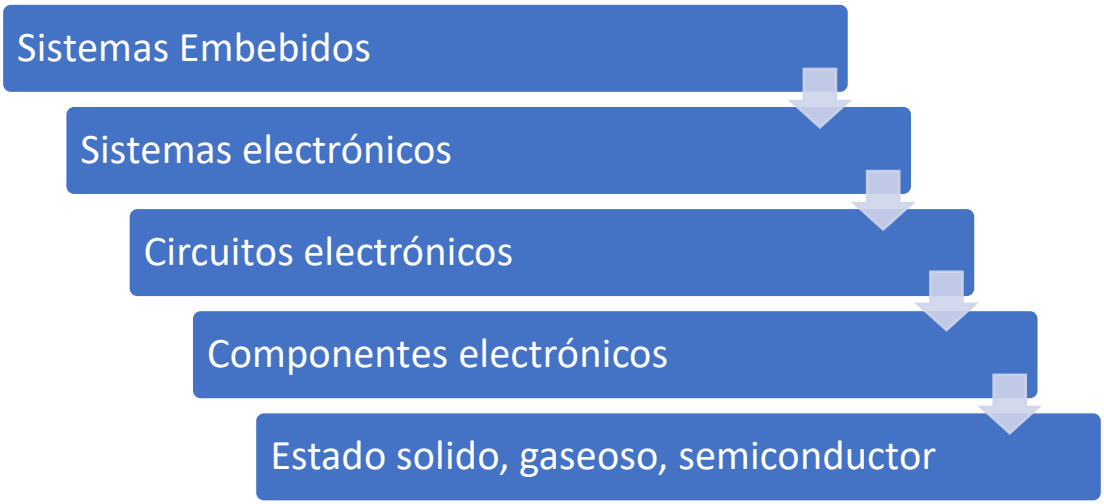
La tercer sección, corresponde a los componentes electrónicos que se pueden llegar a utilizar. El botón de la derecha al combo de componentes (con forma de lista) cambia la vista a una versión en donde se detallan cada uno de los componentes. Cabe aclarar que al ser una simulación, configuraciones y valoraciones que perite realizar sobre los componentes son ideales. En la vida real todos los componentes eléctricos poseen un umbral de trabajo (± su valor). Incluso existen niveles de tolerancia, mientras más preciso sea el componente (menor tolerancia) el componente es más caro (ya que su construcción es más compleja).

4. Electrónica

La **electrónica** es una rama de la física que estudia los movimientos de los electrones libres causados por cargas eléctricas.

La **electricidad** es un conjunto de fenómenos producidos por el movimiento e interacción entre las cargas eléctricas positivas y negativas de los cuerpos físicos conductores. Esta electricidad, existirá mientras no se alcance una compensación entre las cargas de los polos del conductor. La palabra "electricidad" procede del latín *electrum*, y a su vez del griego *élektron*, o ámbar. La energía producida por las cargas eléctricas puede manifestarse dentro de cuatro ámbitos: físico, luminoso, mecánico y térmico. Un sensor es un componente que realiza el proceso inverso, transforma alguno de los elementos físicos descriptos anteriormente en pulsos eléctricos.

Los **sistemas embebidos** funcionan con **electricidad** y utilizan los conceptos de la electrónica fundamental, la electrónica aplicada, y de la ingeniería electrónica para poder funcionar. Por lo que este apunte busca resumir estos conocimientos para facilitar la construcción de sistemas embebidos en *TinkerCad*. El siguiente diagrama muestra el modelo de capas en la que los sistemas embebidos trabajan.

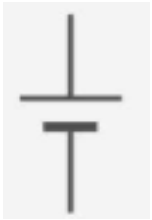


Como primer concepto, la electricidad que necesita para el funcionamiento de los sistemas embebidos es suministrada por un componente llamado fuente de alimentación.

4.1. Fuentes de alimentación

Si entre dos puntos de un conductor no existe diferencia de cargas eléctricas, se dice que ambos puntos posee cero voltaje (lleva una letra “v”, ósea 0V). El Voltaje (recibe su nombre en honor a Alessandro Volta, quien en 1800 inventó la pila voltaica) es la unidad de medición utilizada para medir esa diferencia de cargas eléctricas. También se lo denomina tensión, diferencia de potencial . Ese movimiento de cargas, si se produce en la misma dirección, produce una un flujo de corriente eléctrica, que se mide en Amperes, se expresa con la letra “A”(El nombre fue originado por las iniciales de quien es considerado padre de la

electrodinámica, el físico-matemático de origen francés, André-Marie Ampère). Se llama fuente de alimentación al componente responsable de generar ese voltaje con una determinada corriente eléctrica (movimiento de carga). El siguiente cuadro muestra los tipos de componentes disponibles en *TinkerCad*:

				
Símbolo en circuitos	Voltaje fijo 3V Corriente 30mA	Voltaje fijo 9V Corriente 500mA	Voltaje configurable 1,5V Corriente variable 2,4A a 2,2A	Voltaje Configurable Corriente configurable

La importancia en tener en claro las características de la fuente de alimentación, tiene que ver con las consecuencias que pueden producirse por su mala utilización. Si se utilizará un componente en un circuito que requiere de un voltaje mayor, este se puede quemar. También puede pasar, que si se alimenta el circuito con poco amperaje, no alcanzará para cubrir el consumo de todos los componentes (incluso podría llegar a quemarse por forzarlo a trabajar).

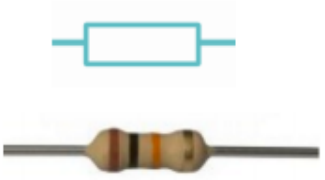
A continuación se hace un repaso por los principales componentes electrónicos que pueden utilizarse en los circuito [1].

4.1. Componentes electrónicos

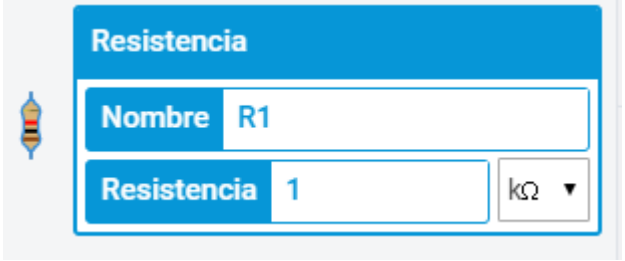
Resistencia:

La función del resistor o resistencia es dificultar el paso de la corriente eléctrica. Como consecuencia, de producir una caída o pérdida de tensión entre sus terminales, permite la bajada en la intensidad del conductor y una transformación de energía eléctrica en calor. En la vida real (nada se pierde). La unidad de medida utilizada es el *ohmio*, en honor al físico George Simon Ohm, y se representa con la letra griega omega (Ω). Para identificar a las resistencias, se las imprime una codificación por colores (en forma de franjas) para identificarlas:

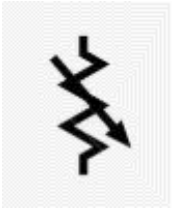
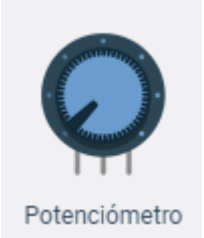
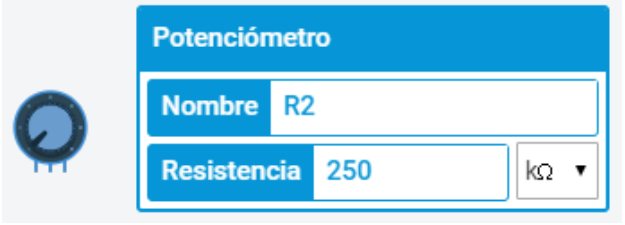
COLOR	CIFRA	CIFRA	MULTIPLICADOR	TOLERANCIA
negro	0	0	x1	
marrón	1	1	x 10	± 1%
rojo	2	2	x 100	± 2%
naranja	3	3	x 1.000	
amarillo	4	4	x10.000	
verde	5	5	x 100.000	± 0,5%
azul	6	6	x 1.000.000	
morado	7	7	x 10.000.000	± 0,1%
gris	8	8	x 100.000.000	
blanco	9	9	x 1000.000.000	
oro			x 0,1	± 5%
plata			x 0,01	± 10%



En el simulador conocer el código de colores no es necesario, ya que puede configurarse el valor de una resistencia desde las propiedades del componente:

Simbolo en circuito	Forma	Tinkercad
		

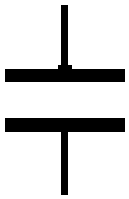
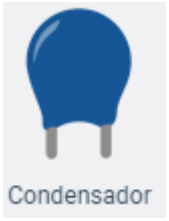
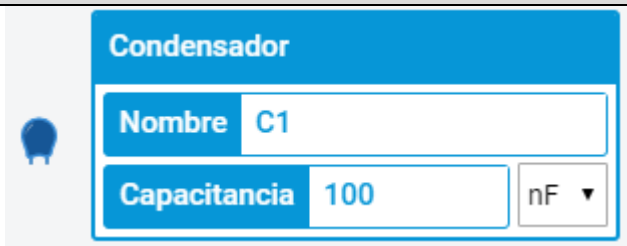
También existen las resistencias variables, conocidas como potenciómetros. Estas permiten interactivamente variar su propiedad resistiva, al hacer pasar la corriente eléctrica a través de una cantidad mayor del material resistivo. Suelen poseer tres pines de conexión, los dos primeros varían la resistencia desde cero al punto de contacto. Mientras que los dos últimos (compartiendo el pin del medio) permiten un valor resistivo que va desde el punto de contacto al máximo valor configurado. En el simulador solo debe configurarse el valor máximo y al ejecutar la simulación del circuito, se permitirá girar el potenciómetro para variar su resistencia.

Simbolo en circuito	Forma	Tinkercad
		

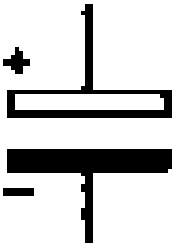
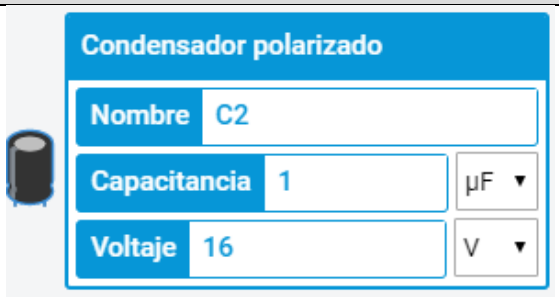
Condensador o Capacitor:

Los componentes condensadores son capaces de almacenar pequeñas cantidades de energía eléctrica. Está formado por dos capas conductoras que se encuentras separadas por un material aislante llamao dieléctrico. La forma de expresar la capacidad es en Faradios “F” (en honor al físico y químico británico Michael Faraday que realizó estudios al electromagnetismo y la electroquímica). Como son tan pequeñas las cargas que almacena se las suele expresar en “η” nano, “μ” micro o “m” mili faradios. Los capacitores pueden ser de dos tipos, los polarizados y los no polarizados. Los primeros, su dieléctrico se construye con láminas de aluminio, papel y óxido de aluminio. Gracias a ello se consiguen mayores capacidades, hay que tener en cuenta su polaridad a la hora de conectarlos en el circuito. Por otro lado se encuentran los capacitores que no tienen polaridad, suelen tener de muy poca capacidad, para su fabricación pueden emplearse dieléctricos como la cerámica, papel o plástico.

Capacitores sin polarizar:

Simbolo en circuito	Forma	Tinkercad
		

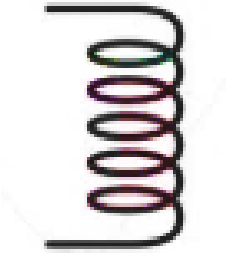
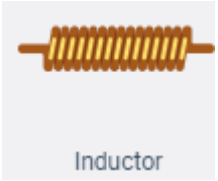
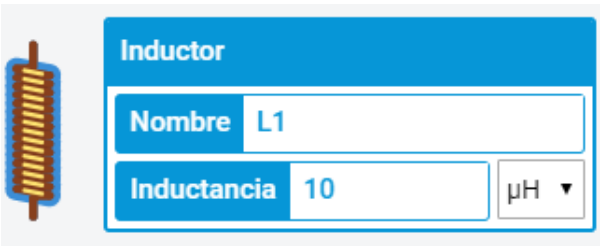
Capacitores polarizados:

Simbolo en circuito	Forma	Tinkercad
		

Nótese que en el simulador deben indicarse la cantidad d Faradios junto con la relación (micrón, nanos, etc). Además de la configuración del voltaje de trabajo, necesario en el capacitor polarizado.

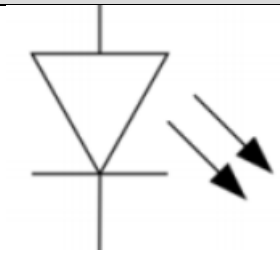
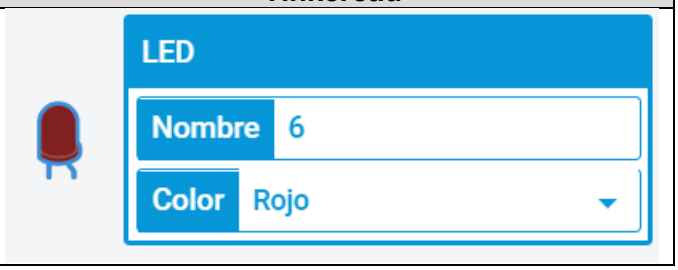
Bobina o inductor:

Es un componente pasivo, que debido al fenómeno de la autoinducción, almacena una pequeña carga en el campo magnético que se forma al pasar electricidad a través de él. Está formado por un hilo conductor envuelto sobre sí mismo en forma de rulo. El su interior se forma un núcleo que puede estar compuesto por aire o material ferroso. Su unidad de medida es el “H” Henrio, Su nombre fue dado en honor del físico estadounidense Joseph Henry. Por la relación entre el campo magnético que genera y el flujo de corriente, su uso básicamente es aplicado en motores, transformadores eléctricos, cerraduras eléctricas, timbres, interruptores deferenciales, circuitos de modulación de señales, entre otros. En el simulador se puede encontrar con el nombre de componente “inductor”:

Simbolo en circuito	Forma	Tinkercad
		

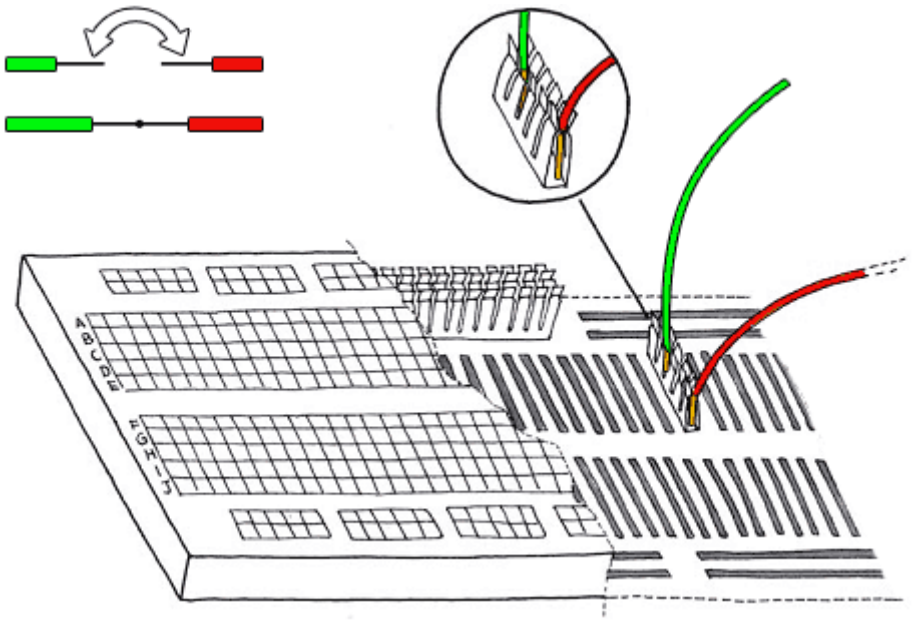
Semiconductor:

Los materiales semiconductores son materiales que se pueden comportar eléctricamente tanto como aislantes como conductores. Dependiendo de diversos factores, por ejemplo: el campo eléctrico o magnético, la presión, la radiación que le incide, o la temperatura del ambiente en el que se encuentre. La palabra semiconductor fue utilizada por primera vez en el año 1782, por Alessandro Volta. Incluso Michael Faraday 1833, documentó la fluctuación de un compuesto dependiendo de la temperatura. El empleo de los materiales semiconductores supuso una revolución en el campo de la electrónica. Componentes como el diodo, el transistor, los sensores y los circuitos integrados, entre otros, se construyen en la actualidad con materiales semiconductores. Por su amplia variedad, por el momento solo comentamos el semiconductor más utilizado el diodo emisor de luz (LED):

Simbolo en circuito	Forma	Tinkercad
		

4.2. Protoboard

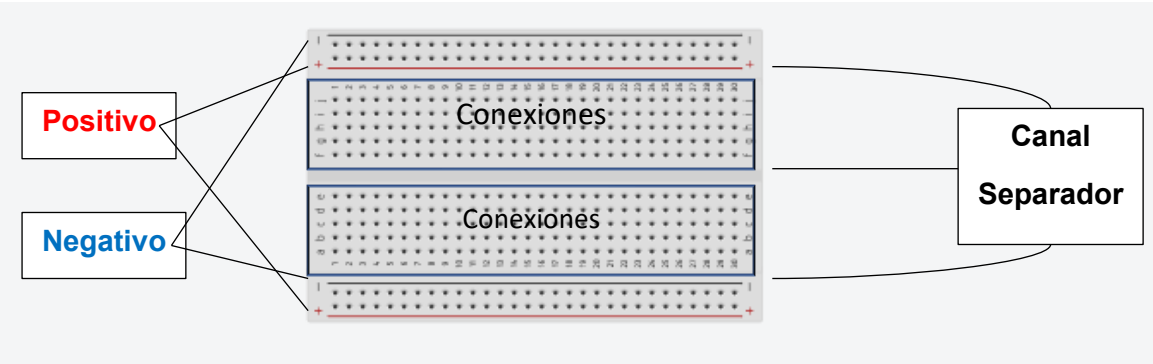
Una placa de pruebas o *protoboard* (su nombre en inglés) es un tablero con orificios que se encuentran conectados internamente siguiendo un patrón. Fue inventada en 1958 por el ingeniero Jack Kilby (es el mismo que aparece en la historia de GPU, inventor de circuito integrado). Su idea es simple, poder aplicar cambios en las reconexiones o reemplazo de componentes rápidamente en los circuitos electrónicos en fase de prototipo. La siguiente imagen muestra la forma como se utiliza la *protoboard* para conectar dos cables, trabajando como puente interno ente los extremos de los cables.



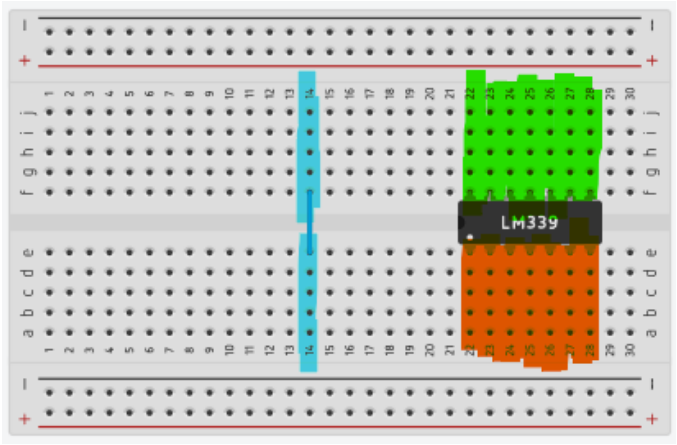
Es utilizada para facilitar el armado de circuitos electrónicos. Ya que otra forma de realizar esto es soldando directamente los componentes a una plaqueta. Dificultando la reconexión o realizar cambios de componentes en el circuito. Otra manera es conectar los componentes atados entre sí. En este caso, al moverse o tocar algún cable, el circuito podría desconectarse. Si bien el simulador permite esta configuración, que los componentes se conecten directamente a los cables, al estar en el simulador no se llegarán a desarmar. Desde la cátedra les pedimos que utilicen a la *protoboard* para tener la experiencia de su uso. Así el día de mañana que realicen un circuito (por ejemplo para la materia proyecto o un trabajo por su cuenta) ya tienen el conocimiento sobre cómo utilizarla.

El simulador posee tres tipos de *protoboard* y la diferencia es el tamaño. Que repercute en su correspondiente cantidad de componentes a conectar:

Hay una forma estándar de utilizarla ya que las conexiones internas corresponde a diferentes usos en distintos bloques. Los canales que posee, son utilizados para separar los bloques. En los extremos de la placa se encuentran los bloques de energía, donde se conecta la fuente de alimentación, con los conectores positivo y negativo. Mientras que en su interior se encuentran los dos bloques de conexiones. Los distintos bloques están formados por pequeños nodos, que son los orificios donde se conectan los componentes.



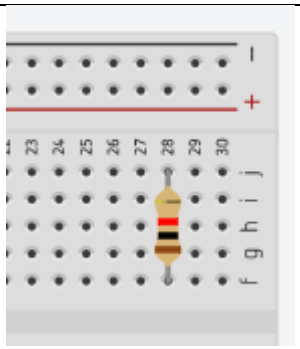
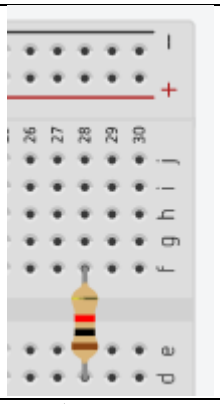
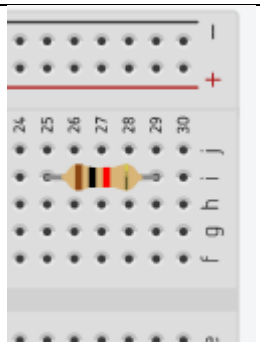
Los bloques de conexión son separados por el canal central, cuya disposición y tamaño permite utilizar determinados componentes con muchos pines, como por ejemplo los circuitos integrados:



El circuito integrado que se encuentra a la derecha de la *protoboard*. Se pueden ver las zonas coloreadas de los extremos del integrado representan secciones en donde los pines de un lado pueden conectarse cada independientemente de otro lado. Mientras que a la izquierda se encuentra un conector (color celeste), que muestra como ambos extremos comparten la misma conexión, ya que el cable funciona como puente uniendo las pistas.

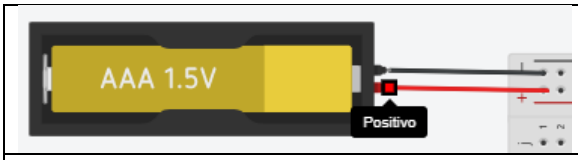
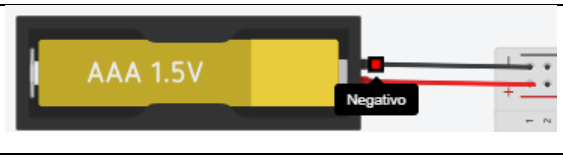
4.3. Guía rápida de conexiones de circuitos eléctricos y Protoboard.

Forma de conectar correctamente un componente en la *protoboard*:

		
✖ Incorrecto	✓ Correcto	✓ Correcto

Respetar código de colores en los conectores:

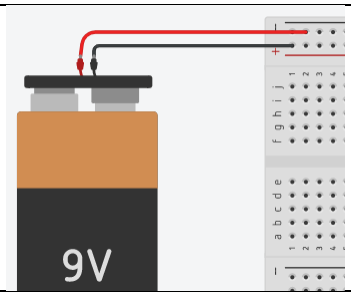
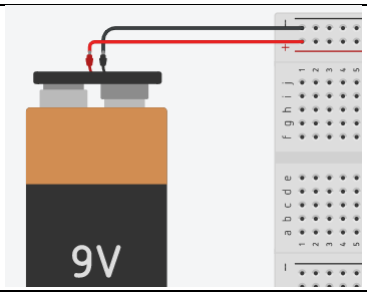
- Conector color Rojo (●): Utilizado para indicar que el cable transporta energía positiva.
- Conector color Negro (●): Utilizado para indicar que se conecta al polo negativo (o corriente neutra).
- Conector color Verde (●): Utilizada por algunos componentes suele ser utilizado para indicar tierra (gnd)
- Conector de otros colores: Indica que transporta otro tipo de energía/señal eléctrica, que no sea las enumeradas anteriormente.

	
Conexión al polo positivo (rojo)	Conexión al polo negativo (negro)

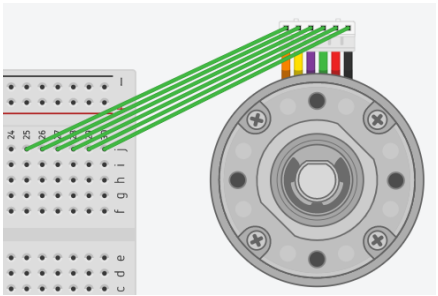
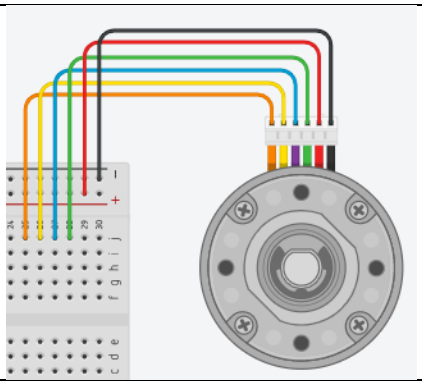
NOTA: *ThinkerCad* es un entorno simulado, por lo que tenemos amplia variedad de colores. Si fuera un prototipo real, probablemente se usarían solo los colores disponibles. Al ser simulado tenemos la ventaja de hacerlo correctamente.

Forma de conectar fuente de alimentación:

La conexión correcta de la fuente de alimentación es desde la salida positiva, con cable de color rojo, en el otro extremo del cable se conecta a la fila positiva (+) de la *protoboard*. Mientras que la salida negativa se conecta, con el cable de color negro, y este se conecta a la fila negativa (-) de la *protoboard*.

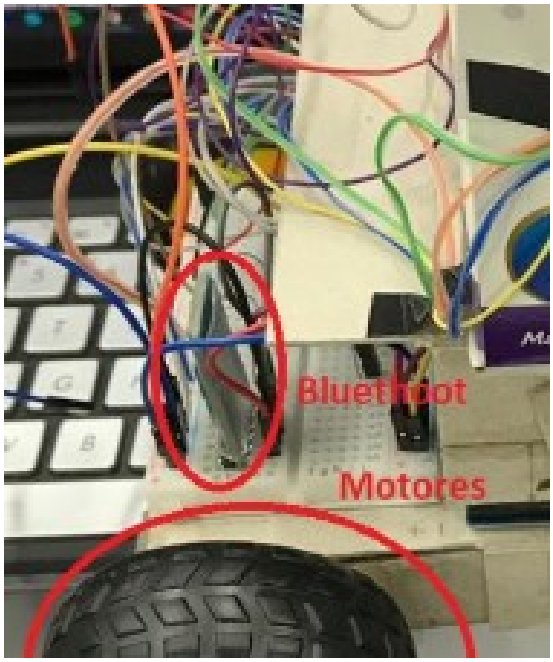
	
✖ Incorrecto	✓ Correcto

Los conectores se distribuyen en forma horizontal y/o vertical (nunca en diagonal):

	
✖ Incorrecto	✓ Correcto

NOTA: También se deben respetar el código de color. Incluso donde se conecta los conectores de alimentación del componente.

La siguiente imagen demuestra como son las conexiones en un prototipo real (ChangoSmart).:



4.4. Leyes de electrónica

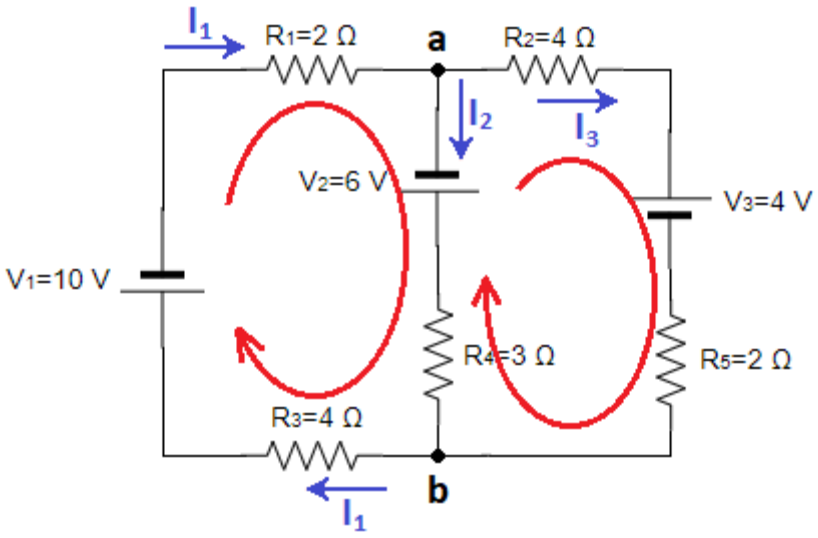
A continuación se enumeran las leyes fundamentales que gobiernan los circuitos electrónicos.

4.4.1. Ley Kirchhoff

La ley Kirchhoff establece que dado un nodo en un circuito, la suma de corrientes entrantes al nodo, debe ser igual a la suma de corrientes saliente de ese nodo.

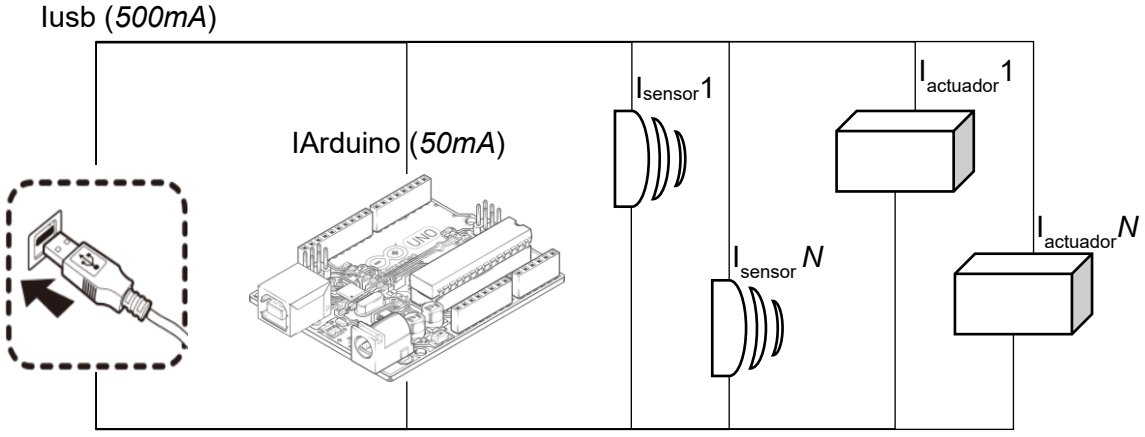
$$I_1 = I_2 + I_3$$

Gráficamente puede verse como, en el “nodo a”, la corriente I_1 se descompone como la suma de las corrientes I_2 e I_3:



Aplicando la ley Kirchhoff en la práctica, representa que Arduino se alimenta por puerto USB, este puede llegar a entregar una corriente eléctrica de hasta 500mA. Esta corriente hay que repartirla entre los componentes internos de Arduino, junto con los sensores y actuadores conectados a él (hasta 40mA cada uno).

$$I_{usb} = I_{arduino} + (\sum I_{sensor}) + (\sum I_{actuador})$$



4.4.2. Ley de Ohm

Establece que la diferencia de potencial (V) aplicada sobre dos nodos es proporcional (R) a la intensidad de la corriente (I). Matemáticamente se muestra de la siguiente forma:

$$V = E \times I$$

En la práctica la ley de Ohm asume que la resistencia R de un camino conductor dado no depende de la corriente (I) o del voltaje (V). Depende de cada componente eléctrico conectado al circuito. Esto es valido siempre a temperatura ambiente normal.

4.4.3. Ley de Watt

Watt descubrió que un componente en funcionamiento consume una cierta cantidad de energía eléctrica. A este fenómeno se lo llama potencia eléctrica (P) y es la relación directamente proporcional a la tensión de alimentación (V) del circuito y a la intensidad de corriente eléctrica (I) que circula por él.

$$P = V \times I$$

En la practica la potencia es la rapidez con la que se realiza un trabajo durante una cantidad de tiempo. Por ejemplo calcular la corriente que consume una lampara de 8w alimentada con 2020v.

4.4.4. Ley de Joule

Establece que cuando una corriente circula por un conductor, encuentra una dificultad que depende de cada material (R), esto produce una perdida de tensión y potencia (Pp), que se da en forma de calor. Matemáticamente se expresa:

$$P_p = R \times I^2$$

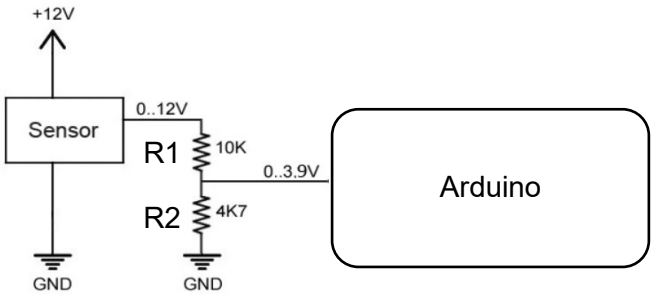
En la practica existen componentes que calientan tanto que para preservar su integridad se les debe agregar una lámina de aluminio como disipador de calor (por ejemplo el l293D usado en el puente H, que controla a los motores por *Arduino*). Incluso pueden llevar un ventilador para ayudar a que circule el aire (usado en microprocesadores). Otros componentes aprovechan esta disipación de calor como es el caso de las pavas eléctricas, secador de pelos, tostadoras, etc. Incluso los fusibles, cuando pasa una gran cantidad de corriente, estos se queman, preservando la integridad del circuito. Otro ejemplo el la luz, por filamento, en el la corriente que pasa por el filamento, logra que se disipe la energía en forma de luz.

4.5. Divisor de tensión

Aplicando las leyes antes vistas, puede deducirse el comportamiento de un divisor de tensión. En este caso el circuito se lo configura para que repartir la alimentación de entrada (Ventrada) entre los *N* nodos (Vsalida) al hacerlo pasar por impedancias (IMP) conectadas en serie. Su fórmula general es:

$$V_{salida} = \frac{IMP_2}{IMP_1 + IMP_2} \times V_{entrada}$$

Como impedancia (IMP) puede utilizarse tanto resistencias o capacitores. Esta configuración es muy útil en sistemas embebidos y en especial Arduino. Ya que Arduino su entrada analógica, solo permite leer valores de 0 a 5 Volts. Para lograr medir sensores que trabajen con mayor voltaje, como por ejemplo el de movimiento que funciona con 12v, se utiliza esta técnica.

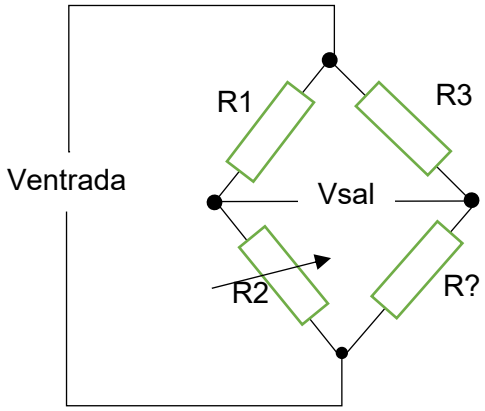


El calculo de las resistencias sigue la ecuación:

$$3.9v = \frac{4{,}7k}{4{,}7k + 10k} * 12v$$

4.1. Puente de Wheatstone

Esta configuración se utiliza para conocer el valor de una resistencia desconocida, utilizando otras 3 resistencias conocidas, como referencia. La configuración del puente se realiza con las resistencias conectadas en dos conjuntos en par de resistencias conectadas en serie. Lo que termina formando son cuatro nodos, dos son utilizados para alimentar el circuito y otros dos para medir la diferencia de tensión obtenida. El siguiente grafico ejemplifica el circuito:



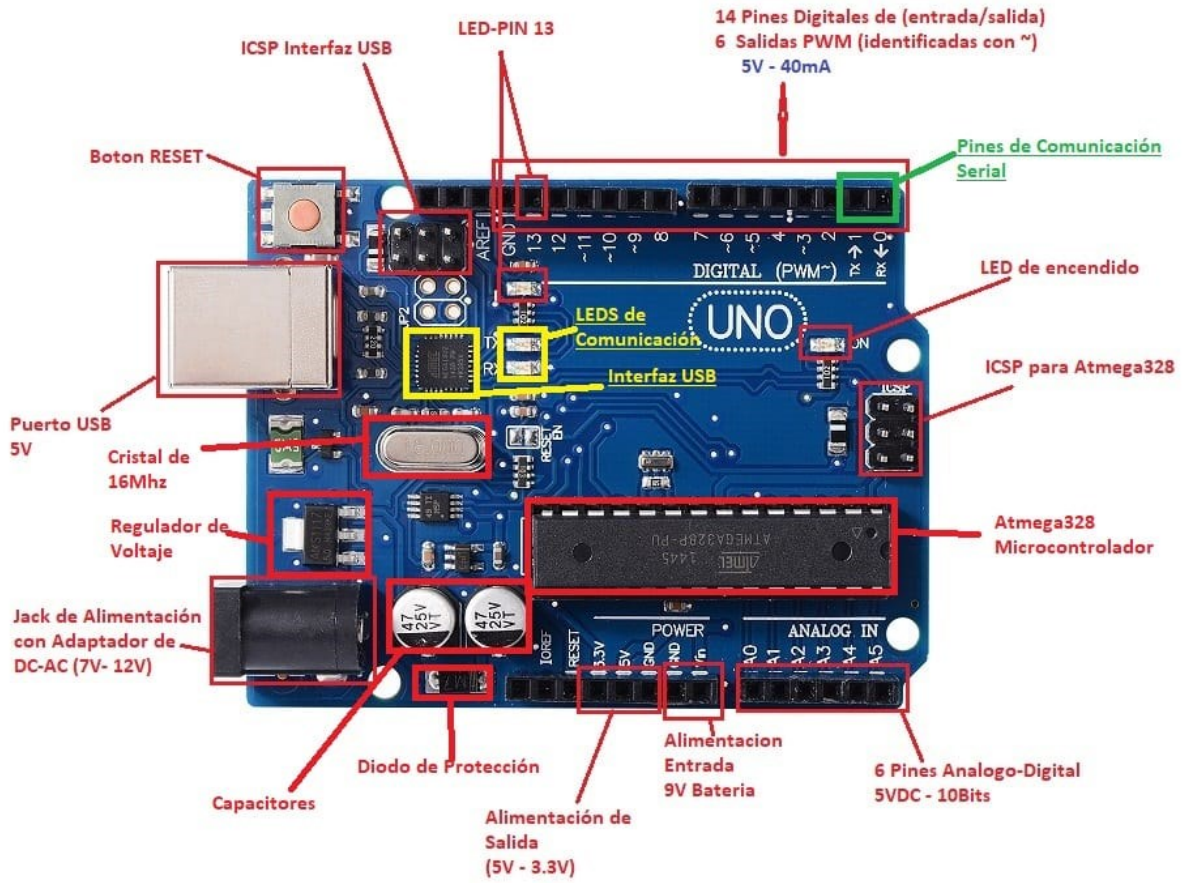
$$V_{salida} = \left(\frac{R_2}{R_1 + R_2}\right) - \left(\frac{R_?}{R_3 + R_?}\right) \times V_{entrada}$$

Nótese que la resistencia R2, es variable. Esto posibilita calibrar el circuito en su condición de reposo o su variación de voltaje ante un evento conocido. Este circuito se suele utilizar para sensores resistivos como las celdas de carga o sensores de temperatura.

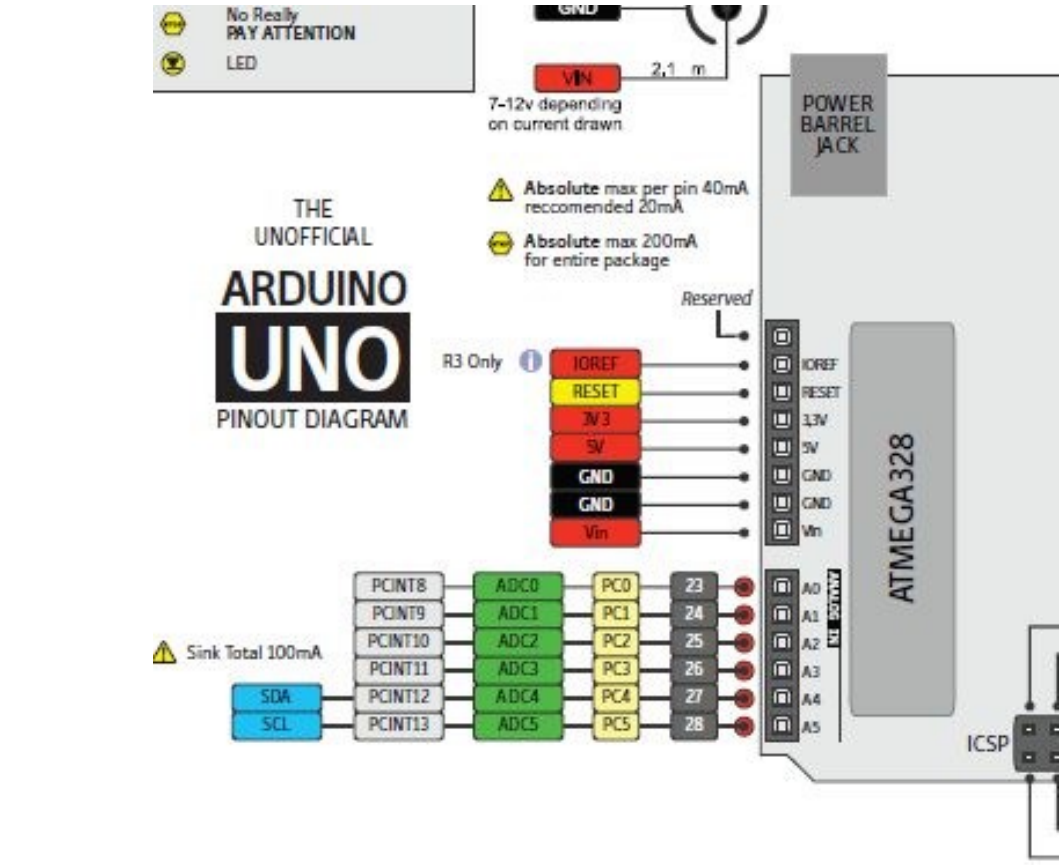
5. Arduino

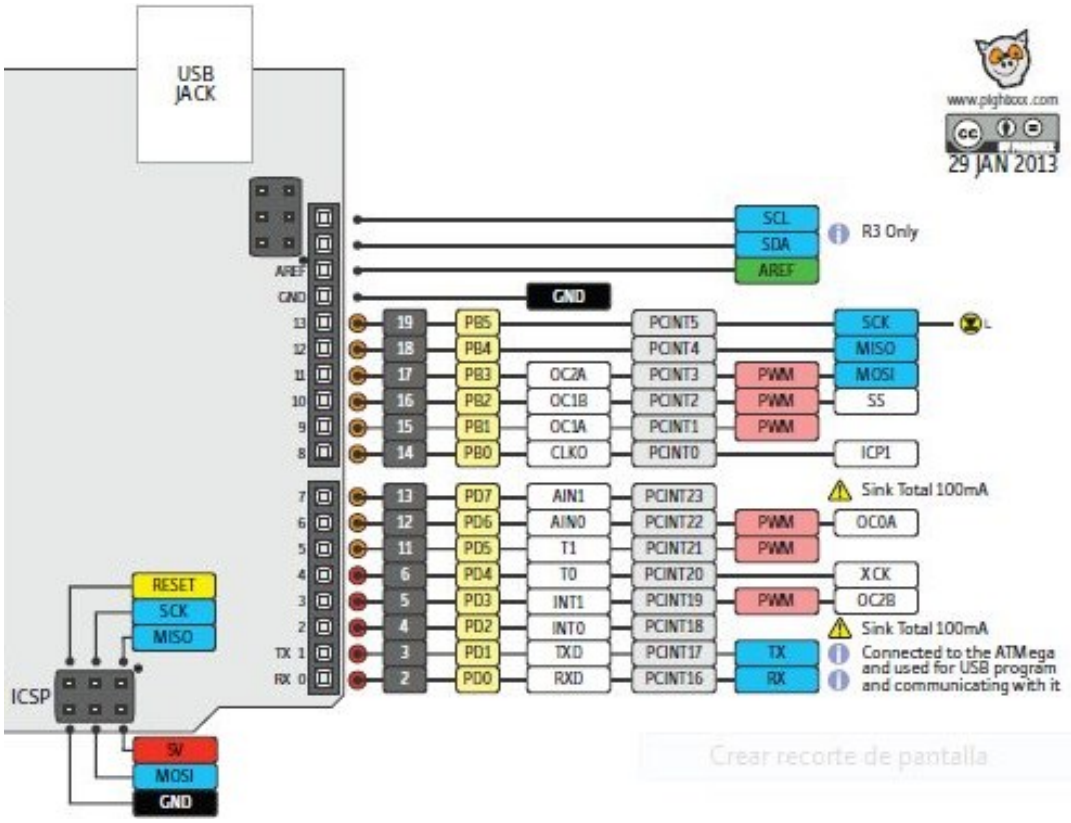
Arduino es un tipo de sistema embebido libre (http://www.arduino.cc), ideado como un equipo de montaje con formato desarrollo. Está compuesto por un microcontrolador AVR, de la marca Atmel (http://www.atmel.com). Cada tipo de microcontrolador define las características y prestaciones, como así la cantidad de puertas digitales y analógicas que puede manejar. El microcontrolador está montado sobre una PCB (*printed circuited board*), que facilita la integración con el resto de componentes (por ejemplo reloj y conectores de alimentación, USB) como así también los pines de conexión a sensores y actuadores. El diseño del hardware Arduino esta inspirado en el hardware libre, *board wriring* (http://www.wiring.co) [2].

5.1. Componentes



5.2. Detalle de conexiones:





5.3. Interrupciones externas

En Arduino, se puede programar para que reconozca interrupciones de eventos externos. La cantidad de interrupciones externas que se pueden detectar dependen del modelo.

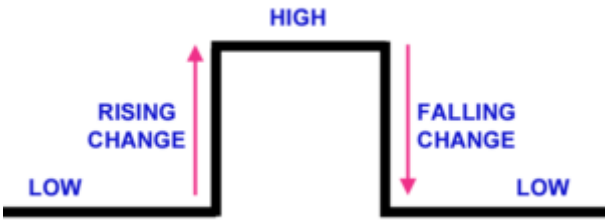
El microcontrolador de Arduino ONE y Nano y MINI poseen 2 pines:

- Pin digital 2 (INT 0).
- Pin digital 3 (INT 1).

Arduino MEGA (microcontrolador Atmega 2560) posee 6 pines:

- Pin digital 21 (INT 0)
- Pin digital 20 (INT 1)
- Pin digital 19 (INT 2)
- Pin digital 18 (INT 3)
- Pin digital 2 (INT 4)
- Pin digital 3 (INT 5)

La clave de su programación es definir en qué forma se dispara la interrupción. Arduino ONE puede detectar 4 formas de estímulos externos, que se definen por como se compone la señal digital:



Estas se definen como :

- LOW: Se invoca al manejador de la interrupción cuando el pin asociado está en estado bajo.
- CHANGE: Se invoca al manejador de la interrupción cuando el pin cambie de valor. Ya sea cuando pase de nivel alto a bajo, o de nivel bajo a alto.
- RISING: Se invoca al manejador de la interrupción cuando el pin cambia del nivel de bajo a alto.
- FALLING: Se invoca al manejador de la interrupción cuando el pin cambia del nivel de alto a bajo.

Existe un quinto estado que solo está disponible en los microcontroladores Arduino Due, Zero y MKR1000 permite:

- HIGH: se lanzará al manejador de la interrupción cuando el pin esté en estado alto.

La forma de asociar la forma que se interpreta el evento con el manejador y la interrupción es utilizando la función attachInterrupt(pin, ISR, modo), que recibe 3 parámetros:

- Pin: Puerto donde el microcontrolador estará detectando la interrupción. Por definición el valor es “0” o “1”, pero se puede utilizar la función interna de writring

digitalPinToInterrupt(puerto), que toma el valor real del puerto y la traduce al pin correspondiente.

- ISR: Función que reemplazara al manejador por defecto, que tiene definido Arduino. Al ser una función de atención de interrupción, esta debe ser lo más simple posible.
- Modo: Es el modo en que se detecta la interrupción y terminara ejecutando el manejador de interrupción.

Seudo ejemplo:

```
#define MODO2 CHANGE   // LOW ,CHANGE, RISING, FALLING.
#define MODO3 CHANGE   // LOW ,CHANGE, RISING, FALLING.
```

```
void setup()
{
   // Toma la primera medición del tiempo.
  attachInterrupt( digitalPinToInterrupt(2), handle_pin2, MODO2 );
  attachInterrupt( digitalPinToInterrupt(3), handle_pin3, MODO3 );
  ...
}
```

```
void loop()
{
  ...
  // Lógica principal.
}
```

```
void handle_pin2( void )
{
  ...
  // Lógica del manejador Pin 2.
}
```

```
void handle_pin3( void )
{
  ...
  // Lógica del manejador Pin 3.
}
```

5.4. Programación del temporizador

La función *delay*() lo que realiza es esperar el tiempo, expresado en mili segundo pasado por parámetro. El uso de esta función es desaconsejada, ya que bloquea la ejecución del programa. Este bloque puede llegar a provocar que se pierdan eventos externos, que el SE debe atender (ejecución sincrónica). Existen dos formas no bloqueantes que permiten reemplazar a la función delay(), programando un temporizador por software o por hardware.

5.4.1. Temporizador por Software

El temporizador por software, consiste en ir midiendo constantemente el tiempo transcurrido, utilizando la técnica de *Polling*. Para luego comparar si la medición realizada llega a superar el umbral de tiempo definido. Si cumple la condición se procede a realizar su acción. Si no cumple la condición, continua el ciclo normal del programa. En Arduino la forma de medir tiempo es utilizando la función **millis()**, que devuelve el tiempo transcurrido expresado en milisegundos.

```
#define TIEMPO_MAX_MILIS 500
unsigned long tiempo_actual,  tiempo_anterior;
```

```
void setup()
{
   // Toma la primera medición del tiempo.
    tiempo_actual = millis();
    ...
}
```

```
void loop()
{
  ...
  // Toma el tiempo actual.
  tiempo_actual = millis();

  // Verifica cuanto transcurrido de tiempo.
  if( (tiempo_actual-tiempo_anterior) > TIEMPO_MAX_MILIS )
  {
    // Actualizo el tiempo anterior.
    tiempo_anterior = tiempo_actual;

    // Realizo la acción...
    ...
  }
}
```

Con esta técnica el programa no se bloquea, se encuentra disponible, ejecutando en ciclo constantemente, para atender eventos externos. Como desventaja, tiene una pequeña degradación en su exactitud, resultado de llamar constantemente a la función millis(), que es notoria con intervalos de tiempo muy pequeños. Otro problema que posee, es que el valor que devuelve la función millis() es limitado (4 bytes) por lo que el contador ascendente se reseteará cada 50 días aproximadamente.

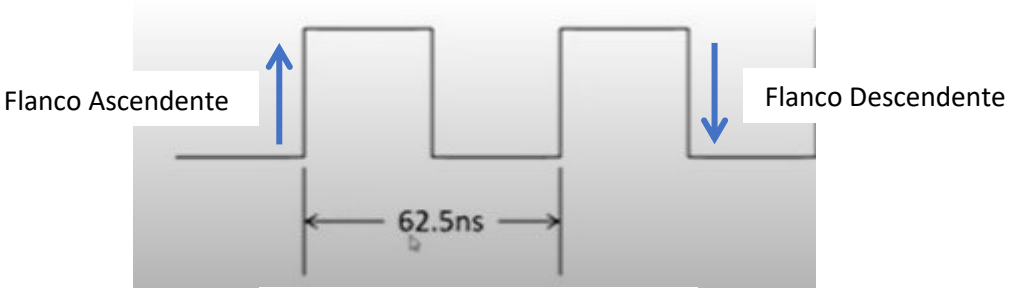
5.4.2. Temporizador por Hardware

Al ser una técnica avanzada de programación en Arduino. Para entender el funcionamiento del temporizador por Hardware, se requiere previamente explicar conceptos teóricos. Haciendo un repaso de nociones sobre las técnicas digitales, los registros temporizadores y las interrupciones.

NOTA: Tener cuidado al usar los temporizadores de hardware. Ya que la biblioteca interna del servo y la función tone() del buzzer, utilizan temporizadores de hardware para funcionar.

Técnicas digitales

El microcontrolador de Arduino funciona a una frecuencia de reloj del sistema que oscila a 16MHz. Esta se representa en forma de onda cuadrada. Su periodo se calcula como 1/16Mhz, o sea 62,5 nano segundos. Gráficamente se puede ver en la imagen de abajo. Por otro lado, el microcontrolador tiene dos formas para detectar el inicio del ciclo. Puede ser por flanco ascendente, cuando la señal pasa de un CERO a UNO lógico. También puede ser, cuando la señal pasa de un UNO a un CERO lógico, flanco descendente.



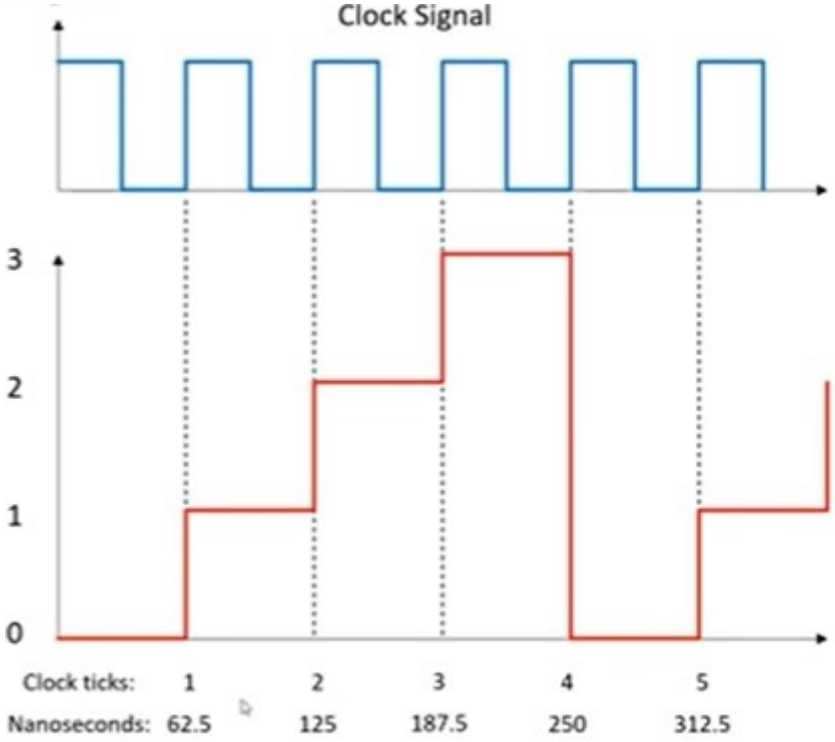
Ciclo de la señal del reloj 16MHz

Registros Temporizadores

La función de un registro contador de tiempo es incrementar su valor a medida que son detectados los pulsos del reloj del sistema. Para explicar su funcionamiento, se define un temporizador de 2 bits. Este va incrementando su valor por cada señal del reloj, cuando supera su límite se resetea empezando nuevamente el conteo.

Valor binario	Valor decimal	Nano Segundos
00	0	0
01	1	62.5
10	2	125
11	3	187.5

Gráficamente puede verse de la siguiente forma:



El microcontrolador de Arduino ONE posee 3 temporizadores:

- TIMER0 (8-bits)
- TIMER1 (16-bits)
- TIMER2 (8-bits)

Que el temporizador cero tenga 8 bits significa que puede configurarse desde el valor 0 o 0b00000000 (binario) al 255 o 0b11111111 (binario).

Arduino MEGA (microcontrolador Atmega 2560) posee 6 temporizadores:

- TIMER0 (8-bits)
- TIMER1 (16-bits)
- TIMER2 (8-bits)
- TIMER3 (16-bits)
- TIMER4 (16-bits)
- TIMER5 (16-bits)

Factor de escala

Factor de escala o divisor de frecuencia suele configurarse en un registro aparte. Sirve para definir un factor multiplicador que actúa sobre temporizador definido. Siguiendo con el ejemplo, se definirá un registro factor de escala de 1 bits, si está activo multiplica por 4 lo definido por el temporizador. Entonces si el factor de escala está activo, cada 4 ciclos de reloj,

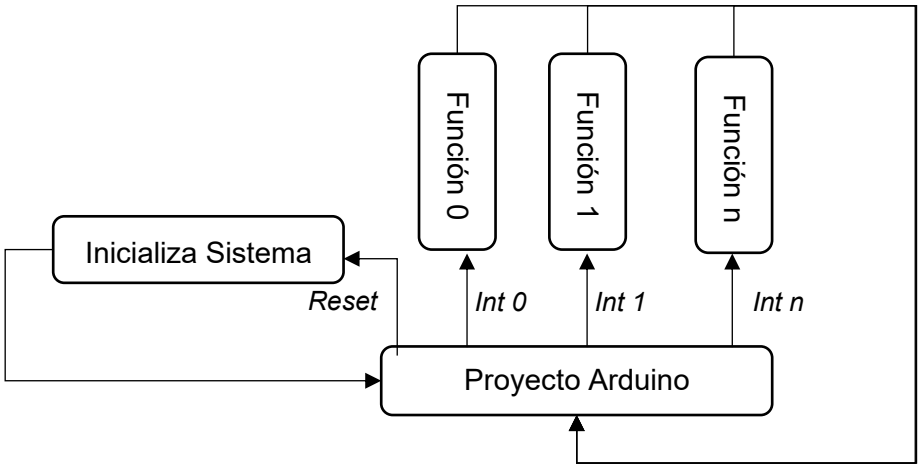
el temporizador incrementara su valor en una unidad (en lugar de hacerlo por cada ciclo). Esto se puede ver de la siguiente forma:

Factor escala (b)	Temporizador (b)	Valor decimal	Tiempo [ns]
0	00	0	0
0	01	1	62.5
0	10	2	125
0	11	3	187.5
1	01	5	250
1	10	6	500
1	11	7	750

Tabla con ejemplo de factor escala 1 bit x 4

Interrupciones.

Las interrupciones es una forma de que el microcontrolador se entere de eventos externos. Esta puede producir por un dispositivo (por ejemplo un sensor) o por un evento periódico producida por una señal digital (previamente configurada). Al controlar los eventos utilizando interrupciones, se evita consumir tiempo del microcontrolador para obtener información del tiempo transcurrido. En Arduino, como cualquier microcontrolador, las interrupciones poseen jerarquía y una forma de ejecución.



La interrupción de *reset* es la única que detiene el programa en ejecución y ejecuta las rutinas de inicialización de sistema, para luego comienza a ejecutar el proyecto desde cero. El resto de las interrupciones, detienen al programa en ejecución, ejecutan la función que le corresponde, que se encuentra definida en el vector de interrupciones. Al terminar la función de interrupción, el programa continuará ejecutando desde donde se detuvo (el punto donde fue interrumpido). A continuación, se encuentra la tabla con el vector de interrupciones de Arduino:

Prioridad	Descripción	Nombre en el vector (ISR)
1	Reset.	
2	Solicitud de interrupción externa 0 (pin D2).	INT0_vect
3	Solicitud de interrupción externa 1 (pin D3).	INT1_vect
4	Solicitud de interrupción por cambio 0 (pins D8 to D13).	PCINT0_vect
5	Solicitud de interrupción por cambio 1 (pins A0 to A5).	PCINT1_vect
6	Solicitud de interrupción por cambio 2 (pins D0 to D7).	PCINT2_vect
7	Interrupción tiempo espera Watchdog.	WDT_vect
8	Temporizador 2 por comparador A.	TIMER2_COMPA_vect
9	Temporizador 2 por comparador B.	TIMER2_COMPB_vect
10	Temporizador 2 por desbordamiento.	TIMER2_OVF_vect
11	Temporizador 1 por captura de evento.	TIMER1_CAPT_vect
12	Temporizador 1 por comparador A.	TIMER1_COMPA_vect
13	Temporizador 1 por comparador B.	TIMER1_COMPB_vect
14	Temporizador 1 por desbordamiento.	TIMER1_OVF_vect
15	Temporizador 0 por comparador A.	TIMER0_COMPA_vect
16	Temporizador 0 por comparador B.	TIMER0_COMPB_vect
17	Temporizador 0 por desbordamiento.	TIMER0_OVF_vect
18	SPI transferencia serial completa.	SPI_STC_vect
19	USART recepción completa.	USART_RX_vect
20	USART Registro de datos vacío.	USART_UDRE_vect
21	USART transmisión completa.	USART_TX_vect
22	ADC Conversión completa.	ADC_vect
23	EEPROM listo.	EE_READY_vect
24	Comparador analógico.	ANALOG_COMP_vect
25	I2C transferencia serial.	TWI_vect
26	Almacenamiento del programa en memoria listo.	SPM_READY_vec

Tabla del vector de interrupciones

En Arduino se puede reprogramar las interrupciones con la función ISR(parametro). Donde “parámetro” es el nombre predefinido de la interrupción, ultima columna de la tabla. En el ejemplo se utilizará “TIMER1_COMPA_vect” que corresponde al temporizador 1 por comparador A.

Programación del Temporizador en Arduino

Ya definida la base teórica, para programar el temporizador de Arduino ONE se utilizan cuatro registros [3]:

- TIMSK: Este registro configurará la interrupción por comparación.
- OCR1A: Permite establecer el valor del contador que permitirá disparar la interrupción.
- TCCR1B: Permite configurar el factor de escala.
- TCCR1A: Funciona en conjunto con TCCR1B, y se utiliza para configurar el modo del temporizador.

Se utilizará un quinto registro, el registro TCNT1 (de 16 bits) que es en donde el microcontrolador va a llevar el conteo de los ciclos de reloj transcurridos para el temporizador 1.

Registro mascara de interrupciones (TIMSK1**)**

El registro TMSK1 (Timer/Counter Interrupt Mask Register 1) se utiliza para habilitar en el temporizador 1, el uso de interrupciones en modo comparador. El registro tiene la siguiente disposición de bits:

7	6	5	4	3	2	1	0	Bits
0	0	ICIE1	0	0	OCIE1B	OCIE1A	TOIE1	**TIMSK1**
-	-	r/w	-	-	r/w	r/w	r/w	

La activación de bits a cada posición corresponde:

- Bit 5 - ICIE1: Se habilita la interrupción cuando se produce la captura del valor que en ese momento tiene el registro TCNT1.
- Bit 2 - OCIE1B: Se habilita la interrupción cuando el registro TCNT1 iguale OCR1B.
- Bit 1 - OCIE1A: Se habilita la interrupción cuando el registro TCNT1 iguale OCR1A.
- Bit 0 - TOIE1: Se habilita la interrupción cuando el registro TCNT1 alcance su valor máximo (65535).

En el ejemplo el TIMSK1 tendrá el valor 2 (que en binario 0b000000010). Significa que la comparación del contador de temporizador (TCNT1) la realizara contra el registro OCR1A.

Control del contador de temporizador 1 A (TCCR1A)

Permite configurar la forma en que se realizará la comparación (OCIE1B o OCIE1A).

7	6	5	4	3	2	1	0	Bits
PWM	PWM	PWM	PWM	0	0	WGM11	WGM10	**TCCR1A**
r/w	r/w	r/w	r/w	-	-	r/w	r/w	

La activación de bits a cada posición corresponde:

- Bit 7-4 : Relacionados con la salida en PWM (no usar “por ahora”).
- BIT 1-0: se usan de la siguiente forma:

Bit 1 - WGM11	Bit 0 - WGM11	Descripción
0	0	OC1A (pin 15) / OC1B (pin 16) desconectado
0	1	OC1A/OC1B desconectado
1	0	Borrado en la comparación, cuenta ascendente
1	1	Borrado en la comparación, cuenta descendente

Para el ejemplo TCCR1A se inicializa en cero, que significa modo normal que OC1A (pin 12) se encuentra desconectado.

Control del contador temporizador 1 B (TCCR1B)

Permite configurar la forma en que se realizará la comparación (OCIE1B o OCIE1A).

7	6	5	4	3	2	1	0	Bits
ICNC1	ICES1	0	WGM13	WGM12	CS12	CS11	CS10	**TCCR1B**
r/w	r/w	-	r/w	r/w	r/w	r/w	r/w	

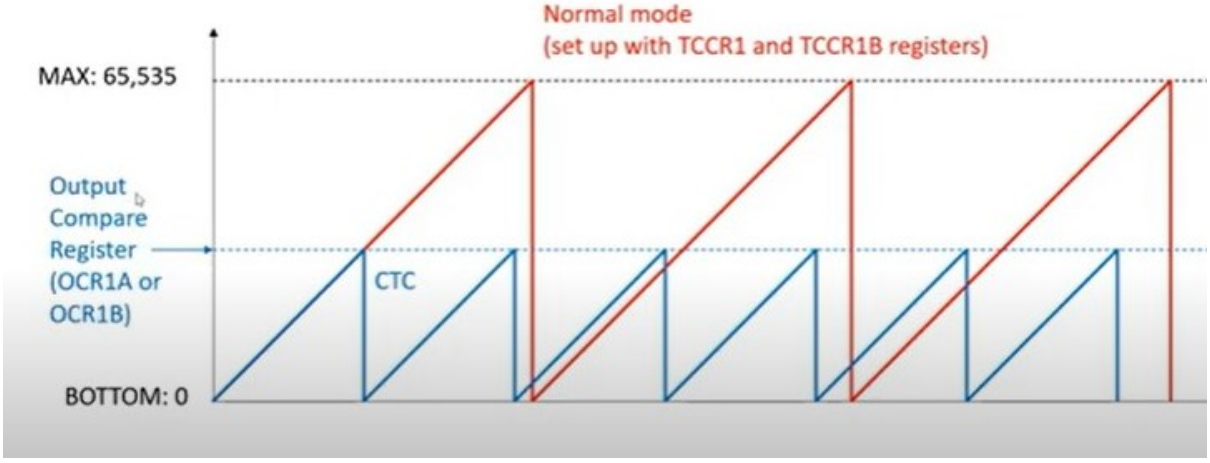
La activación de bits a cada posición corresponde:

- Bit 7-6 : Relacionados con la forma de captura (no usar “por ahora”).
- Bit 4-3: Modo temporizador, ver *Waveform Generation Mode Bit Description* (en datasheet [3]).
- Bit 2-0: Factor de escala del temporizador:

Bit 2 (CS12)	Bit 1 (CS11)	Bit 0 (CS10)	Descripción
0	0	0	Temporizador detenido
0	0	1	x 1
0	1	0	x 8
0	1	1	x 64
1	0	0	x 256
1	0	1	x 1024
1	1	0	Usa el flanco descendente
1	1	1	Usa el flanco ascendente

En el ejemplo el valor del registro TCCR1B es 0b00001101. El cuarto y quinto bits “01” indican el modo de trabajo normal del temporizador. Esto es que resetee el contador del temporizador TCNT1 al llegar al valor del registro OCR1A. Los siguientes tres bits “101”, indican que el factor de escala a utilizar sea por 1024.

El modo del temporizador puede verse gráficamente:



Las líneas coloradas representan al contador en modo normal, cuando TCNT1 llega al valor máximo 65.535, luego se resetea a cero. Las líneas azules, muestra cómo trabaja el modo de comparación, en donde el contador del temporizador TCNT1 se resetea al alcanzar el umbral definido por el registro OCR1A (o OCR1B).

Registro de comparación A (OCR1A) / B (OCR1B)

Ciclo del reloj de sistema es de 16Mhz / 1024 (factor escala configurado 101b) es igual a 15.625Hz o sea 64 microsegundos (1/15.625Hz) . O sea el registro contador TCNT1 va a sumar uno cada 64 us. La ecuación define cuanto tiene que contar TCNT1 hasta llegar a tiempo buscado de interrupción (Ts). Como ejemplo si la interrupción tiene que ser cada segundo la cuenta es:

$$N_{max} = \frac{16\,Mhz \times T_s}{fe} - 1$$

Donde:

- N_{max} : Es el valor a configurar e los registro OCR1A/B.
- T_s: Es el periodo interrupción buscado.
- Fe: es el Factor de escala.

Estos puntos aplicados en el código es:

```
// Se sobre escribe el comportamiento de la interrupción 12
ISR(TIMER1_COMPA_vect)
{
 // Realizo la acción...
}
```

```
void setup()
{
  // El pin OC1A cambia de estado tras la comparación.
  TCCR1A = 0b00000000;

  // Usa el registro OCR1A para comparar y factor escala 8.
  TCCR1B = 0b00001101;

  // Para que la interrupción ocurra al segundo.
  OCR1A  = 15624;

  // Se utilizara la interrupción timer1 por comparación.
  TIMSK1 = bit(OCIE1A);

  // Habilita las interrupciones globales.
  sei();

  ...
}
```

```
void loop()
{
  // Programa principal...
  ...
}
```

6. Bibliografía

[1] L. Prat Viñas, Circuitos y dispositivos electrónicos, Barcelona: Romanyà-Valls, 1999.

[2] Ó. Artero Torrente, ARDUINO, curso práctico, México: Alfaomega, 2013.

[3] Atmel, Datasheet complete, ATmega48PA/88PA/168PA/328P, San Jose, USA: Atmel, 2009.