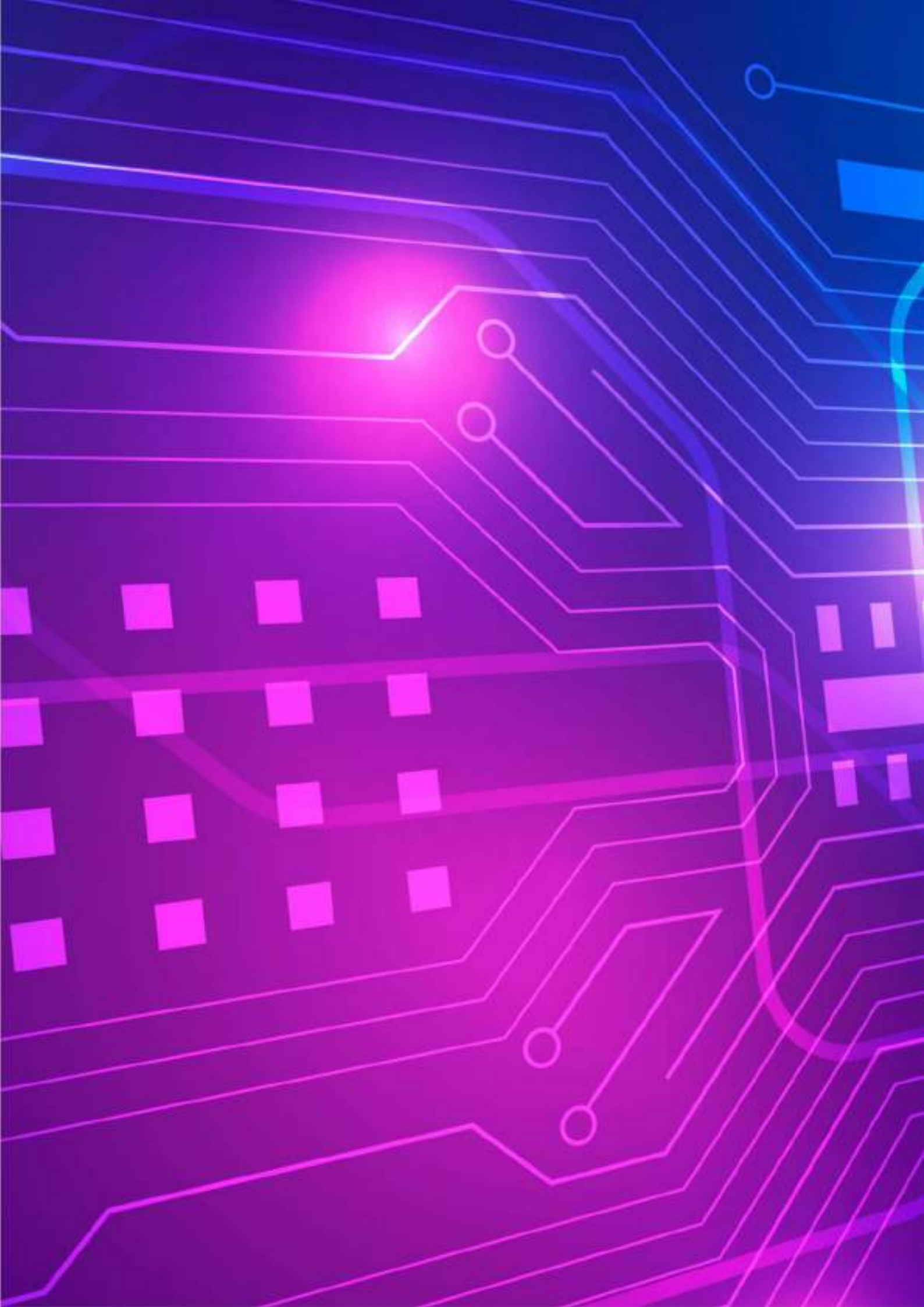
SERIE: TECNOLOGÍA

Electrónica y Automatización de 0-99

Miryam Elizabeth Félix López. Ph.D
César Armando Moreira Zambrano. Ph.D
José Belisario Vera Vera. Mgtr
Freddy Andrés Carrera Sánchez. Mgtr
Yimmy Salvador Loor Vera. Mgtr
Ángel Remigio Cedeño Cedeño
Alexis Xavier Vergara Cedeño




Fondo Editorial
Perspectivas
Globales

Editado y distribuido por

Portoviejo, Manabí, Ecuador. 2024
Instituto de Investigación Multidisciplinaria Perspectivas Globales (IIMPG)
Correo electrónico: fondoeditorial@institutodeinvestigacionperspectivasglobales.org

ISBN: 978-9942-7225-2-2

Serie: Tecnología

ELECTRÓNICA Y AUTOMATIZACIÓN DE 0-99
Autores: © **Miryam Elizabeth Félix López, César Armando Moreira Zambrano, José Belisario Vera Vera, Freddy Andrés Carrera Sánchez, Yimmy Salvador Loor Vera, Ángel Remigio Cedeño Cedeño & Alexis Xavier Vergara Cedeño**

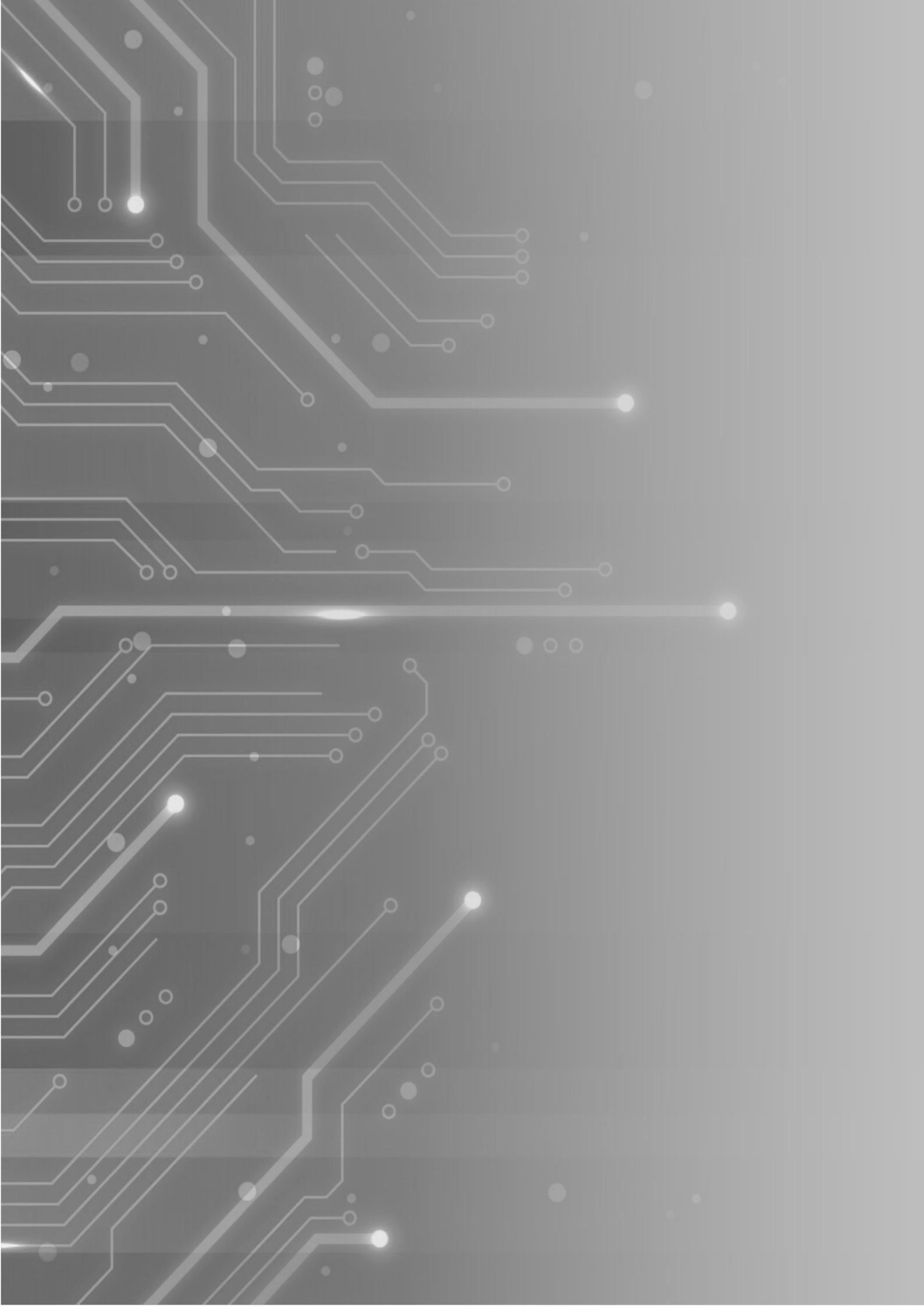
Director del Libro:
Julio Aldana. PhD.

Diseño y diagramación:
Ricardo Díaz (indiolenon@gmail.com)

Revisión, redacción y ortografía:
Carlina García. PhD.

CITAR EL LIBRO

Félix-López, M. E., Moreira-Zambrano, C. A., Vera-Vera, J. B., Carrera-Sánchez, F. A., Loor-Vera, Y. S., Cedeño-Cedeño, Ángel R., & Vergara-Cedeño, A. X. (2024). Electrónica y automatización de 0-99. Fondo Editorial Perspectivas Globales. https://doi.org/10.62574/e1m7yz47



Electrónica y Automatización de 0-99

Miryam Elizabeth Félix López. Ph.D
César Armando Moreira Zambrano. Ph.D
José Belisario Vera Vera. Mgtr
Freddy Andrés Carrera Sánchez. Mgtr
Yimmy Salvador Loor Vera. Mgtr
Ángel Remigio Cedeño Cedeño
Alexis Xavier Vergara Cedeño

SERIE: TECNOLOGÍA

ÍNDICE

CONSTANCIA DE ARBITRAJE

El Fondo Editorial Perspectivas Globales, hace constar que este libro fue sometido a un arbitraje de contenido y forma por jurados especialista en el área de conocimiento de este, mediante el método de doble par ciego.

Realizándose una revisión de contenido de la información, así como del enfoque, paradigma y método investigativo, desde la matriz epistémica asumida por los autores, garantizando así la cientificidad de la obra.

Comité –Editorial “Ad –Hoc” del Fondo Editorial

Fondo Editorial Perspectivas Globales

Libros
Instrumento de evaluación por pares

Indicador	Puntuación			Comentarios
	1	2	3	
1. Título del trabajo			x	
2. Resumen (estructura, coherencia, cumple las normas)			x	
3. Descriptores en función de las normas			x	
4. Relevancia del tema			x	
5. Introducción (rigor argumentativo)			x	
6. Objetivo del articulo			x	
7. Redacción			x	
8. Método (rigor metodológico)			x	
9. Población			x	
10. Instrumento de medición			x	
11. Análisis de la información			x	
12. Calidad de los resultados presentados			x	
13. Capacidad argumentativa en la discusión			x	
14. Contrasta con teoría o autores, destaca los hallazgos de investigación		x		
15. Capacidad argumentativa en la conclusión			x	
16. El trabajo presenta nuevos aportes		x		
17. Cumple con lo establecido en las normas editoriales sobre referencias			x	
Total				

Aprobado = 35 a 51 puntos
Aprobado con modificaciones = 21 a 34 puntos
Reprobado = 1 a 20 puntos

Veredicto por parte del evaluador 1 = Aprobado para su publicación

Fondo Editorial Perspectivas Globales

Libros
Instrumento de evaluación por pares

Indicador	Puntuación			Comentarios
	1	2	3	
1. Título del trabajo			x	
2. Resumen (estructura, coherencia, cumple las normas)			x	
3. Descriptores en función de las normas			x	
4. Relevancia del tema			x	
5. Introducción (rigor argumentativo)			x	
6. Objetivo del articulo			x	
7. Redacción			x	
8. Método (rigor metodológico)			x	
9. Población			x	
10. Instrumento de medición			x	
11. Análisis de la información			x	
12. Calidad de los resultados presentados			x	
13. Capacidad argumentativa en la discusión			x	
14. Contrasta con teoría o autores, destaca los hallazgos de investigación		x		
15. Capacidad argumentativa en la conclusión			x	
16. El trabajo presenta nuevos aportes		x		
17. Cumple con lo establecido en las normas editoriales sobre referencias			x	
Total				

Aprobado = 35 a 51 puntos
Aprobado con modificaciones = 21 a 34 puntos
Reprobado = 1 a 20 puntos

Veredicto por parte del evaluador 2 = Aprobado para su publicación

INTRODUCCIÓN

La electrónica y la automatización son disciplinas fundamentales que han impulsado el desarrollo tecnológico en una amplia variedad de campos, desde la industria manufacturera hasta la electrónica de consumo y la robótica avanzada. En un mundo cada vez más interconectado, el conocimiento profundo de estos sistemas permite a los profesionales enfrentar desafíos complejos y ofrecer soluciones innovadoras que optimizan procesos, mejoran la eficiencia y permiten un control preciso en entornos industriales y domésticos. Este libro, *Electrónica y Automatización de 0-99*, ha sido diseñado para guiar a los lectores a través de un recorrido integral, desde los conceptos básicos hasta los más avanzados, ofreciendo un enfoque práctico y aplicado que facilita la comprensión.

A lo largo de los capítulos, el lector se encontrará con un enfoque claro y didáctico que abarca desde los componentes electrónicos fundamentales como resistencias, condensadores y transistores, hasta el análisis de circuitos digitales y sistemas automatizados. La estructura modular del libro permite que los temas sean abordados de manera gradual, permitiendo a los principiantes adquirir una base sólida en electrónica y a los más avanzados profundizar en temas de automatización, control y programación de microcontroladores. De esta forma, el texto se adapta a diferentes niveles de conocimientos previos.

Una de las características clave de este libro es su enfoque en la práctica. A través de ejemplos, ejercicios y proyectos, el lector podrá aplicar los conocimientos adquiridos en contextos reales. La simulación y el diseño de circuitos con herramientas tecnológicas modernas, como programas de simulación y plataformas como Arduino, se integran en el contenido para asegurar que la teoría se vea reflejada en soluciones aplicables. Este enfoque práctico facilita que los lectores comprendan cómo funcionan los sistemas en la vida real y cómo pueden intervenir para mejorar procesos o desarrollar nuevos dispositivos.

Además, el libro no solo se enfoca en la electrónica básica, sino que también profundiza en áreas clave de la automatización industrial. Desde sistemas de control secuencial y combinacional hasta la implementación de redes de sensores y actuadores, el texto guía a los lectores a través del desarrollo de sistemas inteligentes. Se presta especial atención al diseño de sistemas embebidos, permitiendo que los lectores aprendan a integrar hardware y software de manera efectiva, lo que es crucial en un mundo donde la automatización es cada vez más demandada.

En definición la *Electrónica y Automatización de 0-99* es un recurso completo y actualizado para aquellos que buscan no solo aprender los fundamentos, sino también desarrollar habilidades prácticas que puedan ser aplicadas en contextos reales. Está dirigido tanto a estudiantes como a profesionales que deseen reforzar su conocimiento en electrónica y automatización, dos campos que seguirán siendo claves en el futuro del desarrollo tecnológico global.

DEDICATORIA

Dedicamos este libro a todos aquellos curiosos e incansables aprendices que, con pasión por la electrónica y la automatización, buscan entender y transformar el mundo a través de la tecnología. A los estudiantes que enfrentan los desafíos del aprendizaje con dedicación y esfuerzo, y a los profesionales que siguen impulsando la innovación con sus ideas y habilidades. A la ESPAM MFL por permitirnos desde sus aulas de clases perder aportar a el desarrollo formativo de los estudiantes.

UNIDAD 01

Fundamentos de Sistemas Electrónicos

Fundamentos de Sistemas Electrónicos

TEMA 1:
DISPOSITIVOS ELECTRÓNICOS BÁSICOS

Introducción

No es de sorprender que el comportamiento de un circuito eléctrico dependa del comportamiento individual de los elementos de circuito que comprenden el circuito. Desde luego, diferentes tipos de elementos de circuitos se comportan de manera diferente. Las ecuaciones que describen el comportamiento de los diversos tipos de elementos de circuito se denominan ecuaciones constitutivas. Con frecuencia las ecuaciones constitutivas describen una relación entre la corriente y el voltaje del elemento. La ley de Ohm (V=I*R) es un ejemplo claro de una ecuación constitutiva. (Dorf & Svoboda, 2018), entre estos elementos los más comunes son:

- Resistencias
- Capacitores
- Semiconductores
- Fuentes independientes y dependientes de voltaje y corriente
- Circuitos abiertos y circuitos cerrados
- Voltímetros y amperímetros
- Transductores
- Interruptores
- Etc.

Elementos activos y pasivos

Podemos clasificar los elementos del circuito en dos categorías, pasivos y activos, determinando si absorben energía o suministran energía. Se dice que un elemento es pasivo si la energía total que le llega del resto del circuito es siempre no negativa (cero o positiva). Entonces, para un elemento pasivo, con la corriente fluyendo en el terminal + como se muestra en la Figura 1(a), este consume una energía (w) mayor o igual a 0 ($w >= 0$) en cualquier instante del tiempo.

DATOS ÚTILES

Un **elemento pasivo**, siempre consume energía.

Entonces si el elemento es pasivo porque consume energía, un elemento sería considerado activo si este es capaz de proporcionar energía. Por consiguiente, un elemento activo no satisface la ecuación *w* >=0, cuando se representa por la Figura 1(a). En otras palabras, un elemento activo es aquel que es capaz de generar energía. Los elementos activos son fuentes potenciales de energía, mientras que los elementos pasivos son reductores o absorbedores de energía. Las baterías y los generadores son ejemplos de elementos activos. Considere el elemento que se muestra en la Figura 1(b). Observe que la corriente fluye hacia la terminal negativa y sale de la terminal positiva. Entonces, un elemento es activo si *w* >= 0 en al menos un instante en el tiempo.

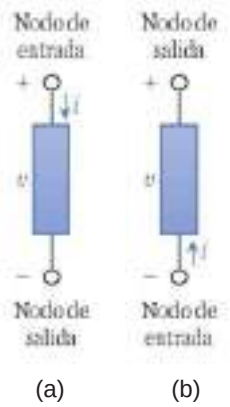


Figura 1: (a) El nodo de entrada de la corriente i es el nodo positivo del voltaje v; (b) el nodo de entrada de la corriente i es el nodo negativo del voltaje v. La corriente fluye desde el nodo de entrada hasta el nodo de salida.

Por otra parte, los elementos eléctricos y electrónicos son aquellos elementos que permiten controlar, cambiar, limitar, modificar, etc., los valores de corriente y voltaje que van a circular dentro del circuito, y para su estudio, es necesario clasificarlos en estas dos categorías: elementos activos y elementos pasivos.

DATOS ÚTILES

Un **elemento activo**, es capaz de suministrar energía.

Para entendernos mejor, los *elementos pasivos* son aquellos elementos que influyen en el flujo de la corriente y las variaciones del voltaje del circuito por su naturaleza de construcción y sólo puede recibir energía, que puede disipar o absorber, entre ellos tenemos: resistencias, bobinas, capacitores, entre otros; mientras que los *elementos activos* modifican el comportamiento del circuito porque tienen la capacidad de controlar eléctricamente el flujo de carga o por una estimulación externa que cambia el comportamiento de su estructura interna de construcción, en este conjunto de elementos podemos encontrar a los transistores, MOSFET, JFET, etc.; y son usados más en la electrónica.

Figura 2:
Elementos pasivos

Figura 3:
Elementos activos

Resistencias

Por concepto básico las resistencias son aquellos objetos que con la capacidad de aguantar u oponerse a otro objeto, la resistencia eléctrica no está alejada de este concepto, porque es un elemento pasivo (consume energía), con la particularidad de oponerse a la corriente eléctrica (I) y de limitar su flujo o paso

dentro del circuito, su valor nominal es medido en ohmios y su símbolo es la letra griega *omega* mayúscula (Ω) y su valor indica la cantidad de resistencia que este elemento posee.

Las resistencias pueden encontrarse en varias presentaciones por su fabricación, en estas tenemos la de fabricación de tipo axial, de montaje superficial y los potenciómetros.

(a) Resistencias axiales (b) Resistencia de montaje superficial (c) Potenciómetros

Figura 4: Tipos de resistencias por su construcción

Como se habló previamente la unidad de resistencia eléctrica es el Ω y para saber el valor en ohmios de una resistencia se dispone de una codificación, que dependerá del tipo de empaquetado. En el caso de las resistencias tipo axial se suele utilizar bandas de colores, pueden ser de 4 a 6 bandas para indicar su valor, donde a más bandas, más preciso es el valor de la resistencia, así como la tolerancia o precisión de esta. El valor de los colores utilizado en este código se muestra en la siguiente tabla.

Tabla 1. Código de colores de resistencias

		Valor de la 1ra. cifra significativa	Valor de la 2da. cifra significativa	Multiplicador	Tolerancia	Coeficiente de temperatura
Negro		0	0	1	-	-
Café		1	1	10	± 1%	100 ppm/°C
Rojo		2	2	100	± 2%	50 ppm/°C
Naranja		3	3	1.000	-	15 ppm/°C
Amarillo		4	4	10.000	± 4%	25 ppm/°C
Verde		5	5	100.000	± 0,5%	20 ppm/°C
Azul		6	6	1.000.000	± 0,25%	10 ppm/°C
Morado		7	7	10.000.000	± 0,1%	5 ppm/°C
Gris		8	8	100.000.000	± 0,05%	1 ppm/°C
Blanco		9	9	1.000.000.000	-	-
Dorado		-	-	0,1	± 5%	-
Plateado		-	-	0,01	± 10%	-
Rosado		-	-	0,001	-	-
Ninguno	-	-	-	-	± 20%	-

Para obtener el valor de la resistencia se obtiene leyendo las bandas que se encuentran en su cuerpo, en el caso de tener 3 o 4 bandas, las dos primeras indican el valor base, mientras que la tercera banda indica el multiplicador de este valor y la cuarta banda nos informa de la tolerancia o precisión de la resistencia. En caso de tener 5 bandas, las tres primeras indicarán el valor base, mientras que la cuarta será el multiplicador, y la quinta la tolerancia.

En la Figura 5, la primera y segunda banda (1ra. y 2da. cifras) corresponden al número 2 por ser rojas, la tercera banda (multiplicador) multiplica el valor de las 2 cifras anteriores por 10.000 (10k) al ser de color amarilla y la cuarta banda (tolerancia), que por ser dorada tiene ± 5%, quedando así: "$[2][2][*10k][\pm 5\%] \rightarrow 220k\Omega \pm 5\%$ $[2][2][*10k][\pm 5\%] \rightarrow 220k\Omega \pm 5\%$".

En la Figura 6, la primera y segunda banda (1ra. y 2da. cifras) corresponden al número 2 por ser rojas, la tercera banda (3ra. cifra) corresponde al 0 por ser negra, la cuarta banda (multiplicador) multiplica el valor de las 3 cifras anteriores por 1.000 (1k) y la quinta banda (tolerancia), que por ser dorada tiene ± 5%, quedando así: "$[2][2][0][*1k][\pm 5\%] \rightarrow 220k\Omega \pm 5\%[2][2][0][*1k][\pm 5\%] \rightarrow 220k\Omega \pm 5\%$".

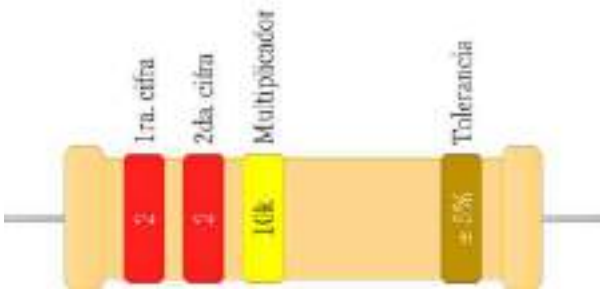


Figura 5: Esquema de código de colores en una resistencia de 4 bandas, a 220 kΩ al ±5% de tolerancia.

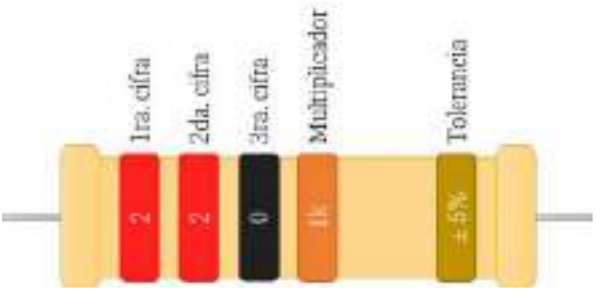


Figura 6: Esquema de código de colores en una resistencia de 5 bandas, a 220 kΩ al ±5% de tolerancia.

En el caso de las resistencias para montaje en superficial, el valor normalmente está impreso en números y los resistores de tolerancia estándar (Standard-tolerance Surface Mount Technology) son marcados con un código de tres dígitos, en el cual los primeros dos dígitos representan los primeros dos dígitos significativos y el tercer dígito representa una potencia de diez (el número de ceros), además, puede llevar la letra 'R', que indica el punto decimal, en caso de ser necesario. A continuación, unos ejemplos:

122	1R6	R22
1ra. Cifra = 1er. número	1ra. Cifra = 1er. número	La "R" indica coma decimal ("0,")
2da. Cifra = 2do. número	La "R" indica coma decimal	2da. Cifra = 2do. número
3ra. Cifra = Multiplicador	3ra. Cifra = 2do. número	3ra. Cifra = 3er. número
En este ejemplo la resistencia tiene un valor de:	En este ejemplo la resistencia tiene un valor de:	En este ejemplo la resistencia tiene un valor de:
$12x10^2$ = 1.200 ohmios = 1,2 kΩ	1,6 ohmios = 1,6 Ω	0,22 ohmios = 0,22 Ω

SABÍAS QUE

A diferencias de otros componentes, las resistencias no tienen un sentido determinado de montaje.

Condensadores

También denominado capacitor (por su nombre en inglés), es un dispositivo pasivo de almacenamiento de electricidad que tiene diversas aplicaciones, por ejemplo: circuitos resonantes, aplicaciones de acoplamiento y derivación, bloqueo de corriente continua, elementos de determinación de frecuencia y temporización, o en supresión transitoria de voltaje, etc. Básicamente un capacitor almacena energía eléctrica mientras este se encuentre energizado a modo de una batería (por lo general pequeñas cantidades y por pequeños periodos de tiempo.

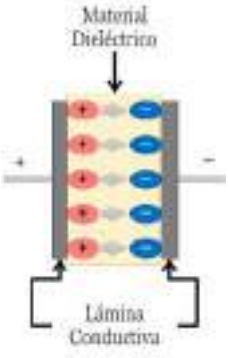


Figura 1: Esquema de un condensador

La construcción básica de un condensador consta de dos láminas o superficies conductivas separadas por una capa de material aislante, como en la Figura 1.

El valor del condensador es medido en Faradios (F) y dependerá del área de las superficies conductivas, de que tan cerca están, y de que tan bueno es el aislante entre ambas. Entre mayor el área de las superficies y menor la distancia, mayor la capacitancia, y viceversa. Al aplicarse una carga positiva sobre una de las superficies, la otra se carga negativamente con la misma magnitud.

Existen varios tipos de condensadores, diferenciados principalmente por el material utilizado en su fabricación. En la Figura 2 mostrada a continuación, de izquierda a derecha, se pueden observar los siguientes tipos: disco de cerámica, de cerámica multicapa (MLC), de electrolito de aluminio, y de electrolito de tantalio. Además, hay otros tipos de condensadores como: los condensadores de vidrio y mica que ofrecen un rango de temperatura de funcionamiento más amplio, pero son más caros.

Figura 2: Algunos tipos de condensadores

La mayoría de los condensadores tiene capacidades por debajo de 1F (el rango típico es de 10pF a 10F). En unos casos, especialmente los de electrolito, la capacidad de capacitancia y el voltaje vienen impresos en el empaquetado. También puede darse el caso de que no traigan ninguna marcación, sino que el

valor viene en el empaque de los componentes. Otros utilizan en código RMK de forma similar a las resistencias de montaje en superficie o SMD, con la diferencia de que el valor base para el cálculo está en picofaradios (pF). Otra diferencia es que no se utiliza la letra 'R' para indicar decimales, sino que se utiliza el prefijo para el valor (Monk, 2017), por ejemplo:

- Un condensador de 100pF tendrá la numeración 101 (10x101 pF = 100 pF).
- Un condensador de 100nF tendrá la numeración 104 (10x104 pF = 100.000 pF = 100 nF).
- Un condensador de 4.7nF tendrá la numeración 4n7.
- Un condensador de 3.3F tendrá la numeración 3F3.

Algunos tipos de condensadores pueden tener polaridad, y se deben conectar adecuadamente para que no se dañen o exploten. Los condensadores polarizados tienen marcado el terminal negativo para evitar errores al instalarlos.

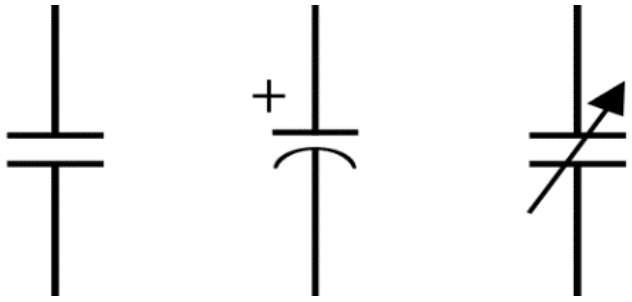


Figura 3: Representación de un capacitor fijo, uno polarizado, y uno variable

SABÍAS QUE

Los condensadores, a diferencia de las resistencias, son menos confiables y pueden fallar (y en algunos casos hasta explotar).

VIDEO

En este video puedes revisar de nuevo cómo funcionan los condensadores
https://www.youtube.com/watch?v=h_m6qFRNITU

Semiconductores

Para iniciar este tema es necesario hablar un poco del átomo y de cómo está construido.

El átomo y el modelo de Bohr

El átomo es la partícula más pequeña de la materia, está compuesta básicamente de un núcleo (protones y neutrones) y de capas (llamada nube de electrones). Los protones (**p**) son partículas subatómicas con una carga eléctrica elemental positiva, los neutrones (**n**) son partículas subatómicas sin carga eléctrica

elemental neta y los electrones (**e**) son partículas subatómicas con una carga eléctrica elemental negativa.

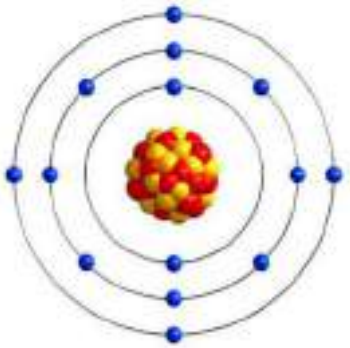


Figura 7: Modelo de Bohr del átomo de silicio (Si), número atómico 14

El modelo de Bohr muestra la estructura del átomo ubicando los electrones en una órbita alrededor del núcleo, este modelo permite cuantificar cuantos electrones (número atómico) posee el átomo y como están distribuidos en las capas (orbitas) que el átomo tiene, ej.: en la Figura. -7 podemos ver que el átomo de silicio (Si por su nomenclatura) 14 electrones ubicados en 3 capas (1ra. capa 2e, 2da. Capa 6e y la 3ra. capa 4e), y este número de electrones es el llamado **número atómico**.

La nube de electrones está constituida por capas electrónicas concéntricas, cuyo número puede variar de 1 a 7, y que se designan con las letras K, L, M, N, O, P y Q. Cada una de estas capas puede tener un máximo número de electrones hasta la 4ta capa cumplen con la formula $2n^2$, donde *n* es el número de la capa: 1ra capa 2e, 2da. capa 8e, 3ra capa 18e, 4ta capa 32e; a partir de la 5ta capa deja de cumplirse porque el máximo registrado en la 5ta. capa es 32e, la 6ta capa 18e y 7ma capa 8e.

El semiconductor

No se puede hablar de semiconductores si antes no presentamos a los elementos conductores y aislantes eléctricos, un elemento es considerado conductor eléctrico, porque su resistencia al paso de la electricidad es muy baja y por un estímulo este puede liberar electrones fácilmente y un elemento es considerado aislante eléctrico cuando bajo ningún estímulo este puede liberar electrones y no permite mover las cargas eléctricas, causando una escasa magnitud de corriente bajo la influencia de un campo eléctrico.

Los elementos o materiales **conductores eléctricos** por excelencia son metales, como el cobre, el oro, el hierro, la plata y el aluminio, y sus aleaciones, aunque existen otros materiales no metálicos que también poseen la propiedad de conducir la electricidad, como el grafito o las disoluciones y soluciones salinas (por ejemplo, el agua del mar). En cambio, para los **aislantes eléctricos** debemos decir que no existe el aislante electico perfecto, porque incluso el mejor de estos permite un ínfimo movimiento de corriente, entre estos tenemos materiales como: plástico, madera, cerámicas, goma, mica, cuarzo, etc.

Por su parte el **semiconductor** es un elemento que por sus propiedades eléctricas puede comportarse como un conductor o un aislante, su comportamiento va a depender de factores como: un campo eléctrico o la temperatura, y sus propiedades pueden modificarse introduciendo “impurezas” en su estructura atómica. Los elementos más utilizados en dispositivos semiconductores son:

- **Silicio (Si)**: Es el elemento más abundante después del oxígeno. Al principio había problemas para su uso. Una vez resueltas, se ha vuelto indispensable para la electrónica moderna. Cuyo número atómico es 14.
- **Germanio (Ge)**: Hasta hace unos años era el único material adecuado para fabricar dispositivos semiconductores. Sin embargo, presentaba problemas y hoy en día ha sido reemplazado por el silicio. Cuyo número atómico es 32.

En electrónica, los elementos conductores, como el cobre, la plata o el oro; suelen ser aquellos que en su estructura atómica tienen un solo electrón en la última capa, denominado **electrón de valencia**, los electrones de valencia son aquellos que dan las propiedades químicas a los átomos, siendo estos los más externos del átomo y por lo tanto alejados del núcleo. La capa donde se alojan es usualmente llamada capa de valencia, en las siguientes figuras es representada en color naranja. En la Figura. -8, tenemos al átomo de cobre, cuyo núcleo consiste en 29 protones y entonces el +29 indica la carga positiva de los 29 protones. El núcleo tiene una carga neta de +1 (ver Figura. -9) que procede de: +29 del núcleo y -28 por los 28 electrones alojados en las 3 capas interiores (2+8+18).

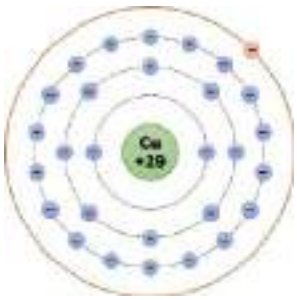


Figura 8:
Átomo de cobre (Cu), número atómico 29.

Figura 9:
Equivalente estructural del átomo de cobre.

Los semiconductores, por otra parte, tienen más electrones en su capa de valencia, por lo que su parte interna tiene una valencia mayor. Por ejemplo, en el caso de átomo de silicio presentado en la Figura 10, la valencia interna es de +4, porque tiene 4 electrones en su capa más externa. Esto hace que sea menos fácil extraer un electrón del átomo, por requerir un poco más de fuerza externa.

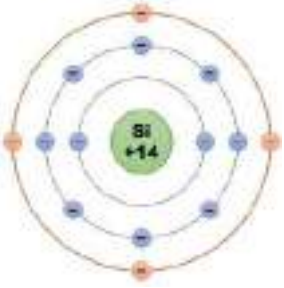
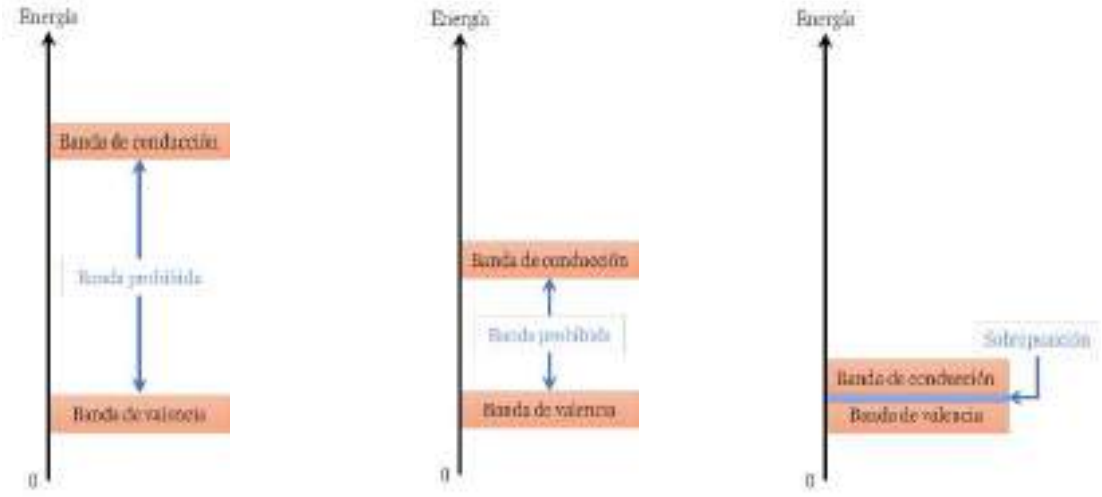


Figura 10:
Átomo de silicio (Si), número atómico 14

Figura 11:
Equivalente estructural del átomo de silicio

Bandas de energía

La capa de valencia de un átomo también representa una banda de niveles de energía y sus electrones están confinados a dicha banda, también llamada **banda de valencia**. Ahora, si un electrón es excitado con un cierto nivel de energía adicional este puede salir de la capa de valencia y convertirse en un **electrón libre**, cuando esto sucede, el electrón libre pasa a lo que se conoce como **banda de conducción**. La diferencia existente entre las bandas de valencia y de conducción se llama banda prohibida. Ésta es la cantidad de energía que un electrón de valencia debe tener para saltar de la banda de valencia a la de conducción. Una vez en la banda de conducción, el electrón es libre de moverse por todo el material y no queda enlazado a ningún átomo dado (Floyd, 2018).



(a) Aislante (b) Semiconductor (c) Conductor

Figura 12: Diagramas de energía para los tres tipos de materiales.

Enlace covalente

Los átomos de silicio se combinan para formar un sólido y lo hacen según un patrón ordenado denominado cristal. Cada átomo de silicio comparte sus electrones con cuatro átomos vecinos, de tal forma que tiene ocho electrones en su capa de valencia. Cuando capa de valencia tiene ocho electrones, esta se satura porque ya no puede entrar ningún otro electrón en esta capa. Cuando el átomo comparte un electrón con otro átomo producen enlaces covalentes y estos son los encargados de mantener unidos a los átomos.

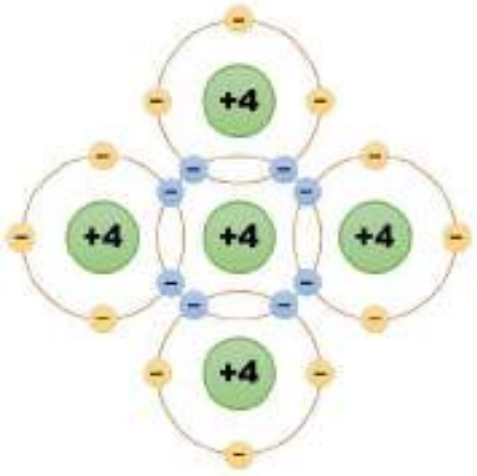


Figura 13: Ilustración de enlaces covalentes de silicio, donde el átomo de silicio central comparte un electrón con cada uno de los cuatro átomos de silicio circundantes, con lo que se crea un enlace covalente con cada uno. Los átomos circundantes están a su vez enlazados con los otros átomos, y así sucesivamente.

SABÍAS QUE

Nadie sabe exactamente por qué el orbital exterior de todos los elementos tiene una predisposición a tener ocho electrones. Cuando de forma natural un elemento no tiene ocho electrones de valencia, este se combina y los comparte con otro átomo.

Un cristal de silicio intrínseco (puro) a temperatura ambiente tiene energía calorífica (térmica) suficiente para que algunos electrones de valencia salten la banda prohibida desde la banda de valencia hasta la banda de conducción, convirtiéndose así en electrones libres, que también se conocen como **electrones de conducción**. Cuando un electrón salta a la banda de conducción, deja un espacio vacío en la banda de valencia dentro del cristal. Este espacio vacío se llama **hueco** (Figura 14). Por cada electrón elevado a la banda de conducción por la energía externa, queda un hueco en la banda de valencia, creando lo que se llama un **par electrón-hueco**. La **recombinación** (Figura 15) se produce cuando un electrón de la banda de conducción pierde energía y vuelve a caer en un hueco en la banda de valencia. En resumen, un trozo de silicio intrínseco que está a temperatura ambiente tiene en cualquier momento, un número de electrones de la banda de conducción (libres) que no están unidos a ningún átomo y que, esencialmente, van a la deriva por todo el material, así también hay un número igual de huecos en la banda de valencia que se crean cuando estos electrones saltan a la banda de conducción (Floyd, 2018). En las figuras a continuación se representan ambos fenómenos.

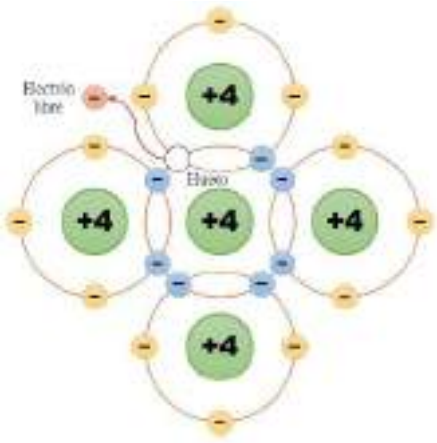


Figura 14:
Hueco generado por electrón liberado.

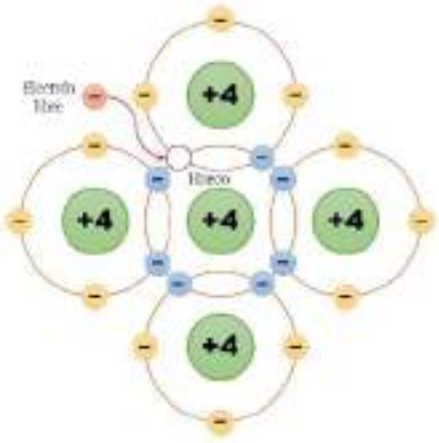


Figura 15:
Recombinación de un hueco y un electrón libre.

Flujo de la corriente de electrones y huecos

Cuando se aplica un voltaje a través de una pieza de silicio intrínseco, como se muestra en la Figura 16, los electrones libres generados térmicamente en la banda de conducción, que son libres de moverse aleatoriamente en la estructura cristalina, son ahora fácilmente atraídos hacia el extremo positivo. Este movimiento de electrones libres es un tipo de corriente en un material semiconductor y se denomina **corriente de electrones**.

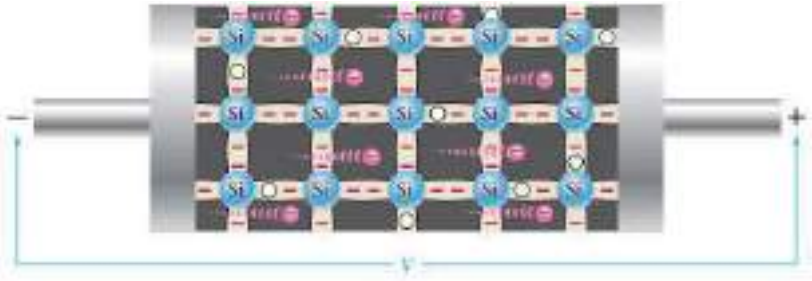


Figura 16: La corriente de electrones en silicio intrínseco
se produce por el movimiento de electrones libres generados térmicamente.

Otro tipo de corriente se produce en la banda de valencia, donde existen los huecos creados por los electrones libres. Los electrones que permanecen en la banda de valencia siguen unidos a sus átomos y no son libres de moverse aleatoriamente en la estructura cristalina como los electrones libres. Sin embargo, un electrón de valencia puede desplazarse a un hueco cercano con poco cambio en su nivel de energía, dejando así otro hueco en su lugar de origen. Efectivamente, el hueco se ha movido de un lugar a otro en la estructura cristalina, como se ilustra en la Figura 1-16. Aunque la corriente en la banda de valencia es producida por los electrones de valencia, se llama **corriente de huecos** para distinguirla de la corriente de electrones en la banda de conducción.

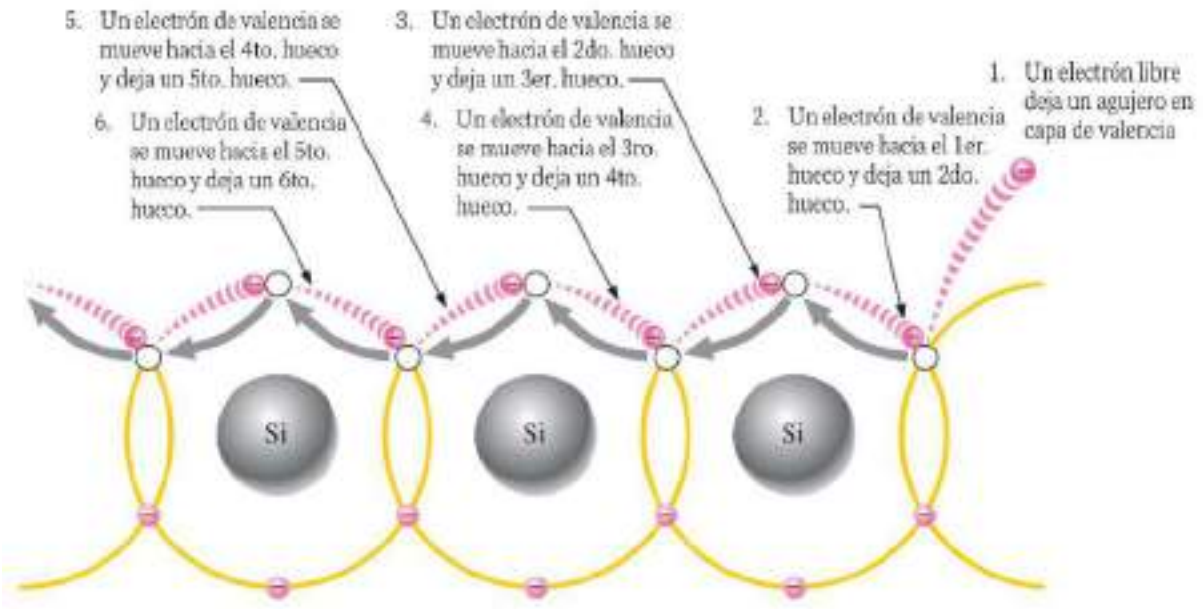


Figura 17: Corriente de huecos en silicio intrínseco.

En la Figura 16 cuando un electrón de valencia se desplaza de izquierda a derecha mientras deja detrás un hueco, éste se ha movido efectivamente de derecha a izquierda. Las flechas gruesas indican el movimiento efectivo de un hueco.

RECUERDA QUE

Los electrones libres y los huecos se mueven en sentidos opuestos en un semiconductor.

Semiconductores Tipo *P* y Tipo *N*

- Un semiconductor intrínseco es un semiconductor puro. Un cristal de silicio es un semiconductor intrínseco si cada átomo del cristal es un átomo de silicio. A temperatura ambiente, actúa como un aislante porque sólo tiene unos pocos electrones libres y huecos producidos por el efecto de la energía térmica.
- Un semiconductor dopado se denomina extrínseco. El dopaje consiste en añadir átomos de impurezas a un cristal intrínseco con el fin de alterar su conductividad eléctrica, especialmente para incrementarla. (Malvino & Bates, 2007)

Dado que los semiconductores son, por lo general, malos conductores, su conductividad se puede aumentar drásticamente mediante la adición controlada de impurezas al material semiconductor intrínseco (puro). Este proceso, llamado **dopaje**, aumenta el número de portadores de corriente (electrones o huecos). Las dos categorías de impurezas son de **tipo *n*** y de **tipo *p***.

Semiconductor tipo N

Para aumentar el número de electrones de la banda de conducción en un elemento intrínseco, se añaden átomos de impurezas pentavalentes. Se trata de átomos con cinco electrones de valencia, como el arsénico (As), fósforo (P), bismuto (Bi) y antimonio (Sb). La *n* significa negativo.

Semiconductor tipo P

Para aumentar el número de huecos en un elemento intrínseco, se añaden átomos de impurezas trivalentes. Se trata de átomos con tres electrones de valencia, como el boro (B), el indio (In) y el galio (Ga). La *p* significa positivo.

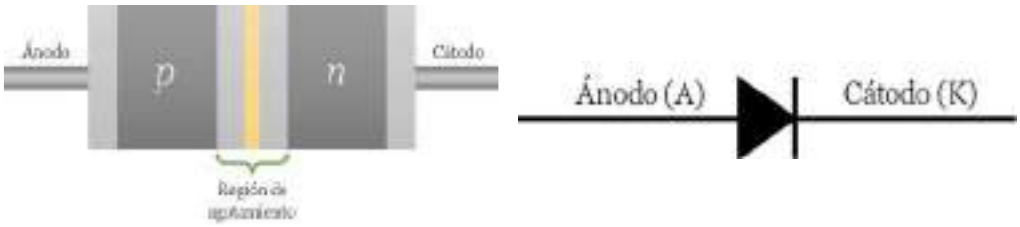
SABÍAS QUE

Para que un fabricante pueda dopar un semiconductor, inicialmente debe fabricarlo como un cristal absolutamente puro. Después, controlando la cantidad de impurezas, pueden controlar de forma precisa las propiedades del semiconductor.

Diodos, funcionamiento y características

El diodo es formado una pequeña pieza de material semiconductor, donde la mitad esta dopada como región *p* y la otra mitad como región *n*, donde la una unión de los materiales tipo *p* y tipo *n*, se la conoce como **juntura *pn*** y una región de agotamiento en medio. La región *p* se llama **ánodo** y está conectada a un terminal conductor. La región *n* se denomina **cátodo** y está conectada a un segundo terminal

conductor. La estructura básica del diodo y su símbolo esquemático se muestran las siguientes figuras.



(a) Estructura básica del diodo (b) Simbología

Figura 18: El diodo.

Funcionamiento de un diodo

Un diodo es un dispositivo electrónico de dos terminales que conduce electricidad en un solo sentido. También se puede decir que es un dispositivo semiconductor que varía sus propiedades de conductividad (conductor o aislante) de acuerdo con la polaridad del voltaje aplicado. Las dos terminales del diodo se denominan ánodo (positivo) y cátodo (negativo) (Graf, 1999).

Dicho de otra manera, un diodo funciona permitiendo el paso de corriente del ánodo al cátodo (polarización directa), pero bloqueando el flujo en sentido contrario (polarización inversa).

Polarización directa (Forward Bias)

Para polarizar un diodo, se aplica una tensión continua a través de él. La polarización hacia adelante o directa es la condición que permite que fluya la corriente a través de la unión *pn*. La Figura **.**-19 muestra una fuente de voltaje de corriente continua conectada por medio de material conductor (contactos y cable) a través de un diodo en la dirección para producir la polarización hacia adelante. Este voltaje o tensión de polarización externa se designa como V_{BIAS}. La resistencia (R_{LIMIT}) limita la corriente en polarización directa a un valor que no dañará el diodo. Observe que el lado negativo de V_{BIAS} está conectado a la región n del diodo y el lado positivo está conectado a la región p. Este es un requisito para la polarización directa o hacia adelante. Un segundo requisito es que el voltaje de polarización, V_{BIAS}, debe ser mayor que el potencial de barrera (V_B).

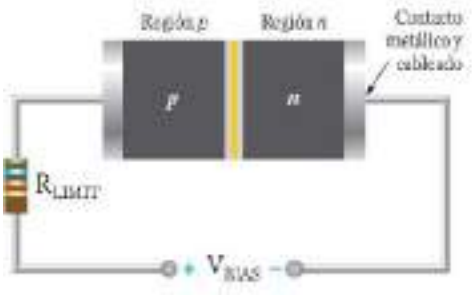


Figura 19: Un diodo conectado en polarización directa.

En la Figura 20 se muestra una imagen fundamental de lo que ocurre cuando un diodo se polariza directamente. Como las cargas similares se repelen, el lado negativo de la fuente de voltaje de polarización “empuja” a los electrones libres, que son los portadores mayoritarios en la región *n*, hacia la unión *pn*. Este flujo de electrones libres se denomina **corriente de electrones**. El lado negativo de la

fuente también proporciona un flujo continuo de electrones a través de la conexión externa (conductor) y hacia la región *n* como se muestra.

La fuente de voltaje de polarización imparte suficiente energía a los electrones libres para que superen el potencial de barrera de la región de agotamiento y pasen a la región *p*. Una vez en la región *p*, estos electrones de conducción han perdido suficiente energía para combinarse inmediatamente con los huecos de la banda de valencia.

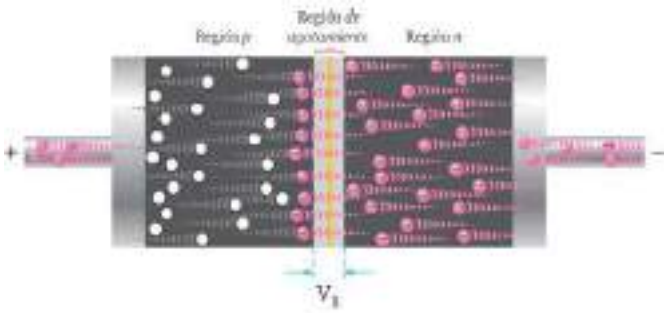


Figura 20: Un diodo polarizado hacia delante que muestra el flujo de portadores mayoritarios y el voltaje debido al potencial de barrera a través de la región de agotamiento.

Ahora, los electrones están en la banda de valencia en la región *p*, simplemente porque han perdido demasiada energía al superar la barrera de potencial para permanecer en la banda de conducción. Como las cargas diferentes se atraen, el lado positivo de la fuente de voltaje de polarización atrae a los electrones de valencia hacia el extremo izquierdo de la región *p*. Los huecos en la región *p* proporcionan el medio o "camino" para que estos electrones de valencia se muevan a través de la región *p*. Los electrones de valencia se mueven de un hueco a otro hacia la izquierda. Los huecos, que son los portadores mayoritarios en la región p, se mueven efectivamente (no realmente) hacia la derecha, hacia la unión, como se puede ver en la Figura 20. Este flujo efectivo de agujeros es la corriente de agujeros. También puedes ver la corriente de huecos como creada por el flujo de electrones de valencia a través de la región p, con los huecos proporcionando el único medio para que estos electrones fluyan.

Como los electrones fluyen fuera de la región *p* a través de la conexión externa (conductor) y hacia el lado positivo de la fuente de voltaje de polarización, dejan agujeros en la región *p*; Al mismo tiempo, estos electrones se convierten en electrones de conducción en el conductor metálico.

Recordemos que la banda de conducción de un conductor se solapa con la banda de valencia, por lo que se necesita mucha menos energía para que un electrón sea un electrón libre en un conductor que en un semiconductor y que los conductores metálicos no tienen agujeros en su estructura. Hay una continua disponibilidad de agujeros que se mueven efectivamente hacia la unión *pn* para combinarse con el flujo continuo de electrones cuando atraviesan la unión hacia la región *p*.

DATOS ÚTILES

Cuando se trabaja con diodos es importante siempre colocar una resistencia que limite la corriente a través de este cuando está polarizado directamente.

- *El efecto de la polarización en directo en la región de agotamiento*
  A medida que los electrones del lado n son empujados a la región de agotamiento, se combinan con los huecos del lado p, reduciendo efectivamente la región de agotamiento. Este proceso durante la polarización directa hace que la región de agotamiento se estreche, como se indica en la Figura 21.
- *El efecto del potencial de barrera durante la polarización directa*
  Recordemos que el campo eléctrico entre los lados positivo y negativo de la unión crea una "colina de energía" que impide que los electrones libres se difundan a través de la unión en equilibrio. Esto crea el potencial de barrera, que en el silicio es de aproximadamente 0,7 V y en el germanio es aproximadamente 0,3 V.

La región de agotamiento se estrecha y se produce una caída de tensión a través de la unión *pn* cuando el diodo se polariza directamente.

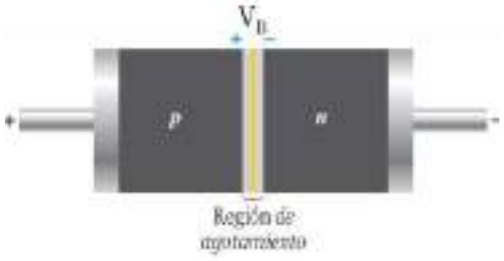


Figura 21:
La polarización directa estrecha la región de agotamiento y produce una caída de voltaje a través de la unión pn igual al potencial de barrera

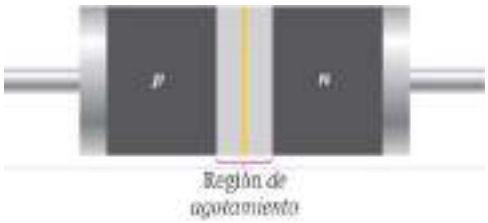


Figura 22:
En equilibrio (sin polarización)

Polarización inversa (Reverse Bias)

La polarización inversa es la condición que esencialmente impide que la corriente viaje a través del diodo. La Figura 23 muestra una fuente de voltaje de corriente continua conectada a través de un diodo en la dirección para producir una polarización inversa. Este voltaje de polarización externa se designa como V_{BIAS} al igual que para la polarización directa. Observe que el lado positivo de V_{BIAS} está conectado a la región *n* del diodo y el lado negativo está conectado a la región *p*. Observe también que la región de agotamiento se muestra mucho más amplia que en polarización directa o en equilibrio.

En la Figura 24 se muestra una ilustración de lo que ocurre cuando un diodo está en polarización inversa. Debido a que las cargas diferentes se atraen, el lado positivo de la fuente de voltaje de polarización "atrae" a los electrones libres, que son los portadores mayoritarios en la región *n*, lejos de la unión *pn*. A medida que los electrones fluyen hacia el lado positivo de la fuente de voltaje, se crean agujeros adicionales en la región de agotamiento. El resultado es un ensanchamiento de la región de agotamiento y menos portadores mayoritarios. Durante la polarización inversa escasos electrones pueden cruzar.

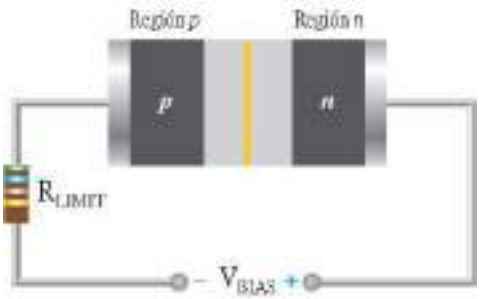


Figura 23: Un diodo conectado con polarización inversa. Se muestra una resistencia limitadora, aunque no es importante en la polarización inversa porque esencialmente no hay corriente.

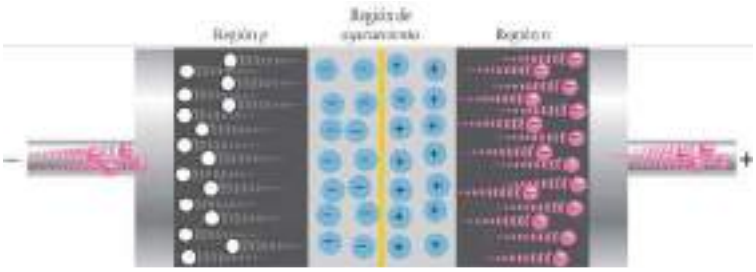
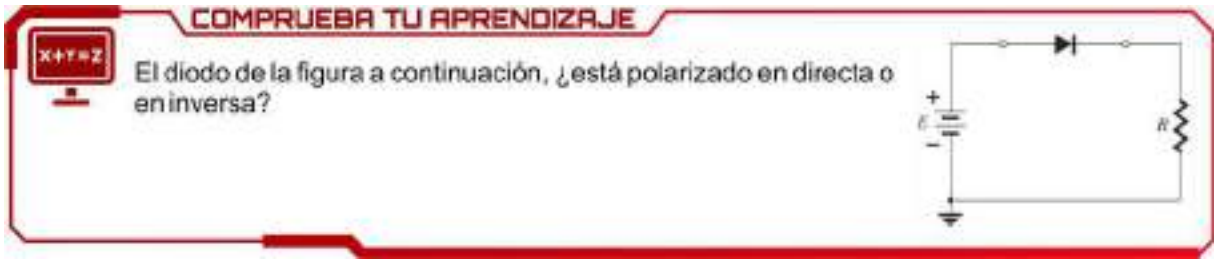


Figura 24: El diodo durante el corto tiempo de transición inmediatamente después de la aplicación de la se aplica la tensión de polarización inversa.



Comportamiento del diodo real y sus aproximaciones (diodo ideal)

Diodo real

A diferencia de otros componentes, que tienen un valor particular marcado o codificado, los diodos están identificados por un código o número del fabricante. Las características de cada diodo, definidas por el fabricante, son al menos: Voltaje de polarización, Corriente máxima, Voltaje de ruptura en DC (V_{BR}), Tiempo de recuperación, etc.

El tiempo de recuperación se refiere al tiempo que le toma al diodo pasar de un estado de bloquear a permitir el flujo de corriente, y suele estar entre los nanosegundos a microsegundos.

El voltaje ruptura en DC es la tensión máxima que puede soportar el diodo en modo bloqueo de corriente. Pasando este valor, el diodo puede permitir el paso de la corriente o dañarse.

Al igual que en el caso de las resistencias, los diodos pueden ser construidos de forma axial o de montaje superficial, aunque también vienen en forma rectangular. A mayor tamaño del diodo, mayor la capacidad de corriente y voltaje que puede soportar.

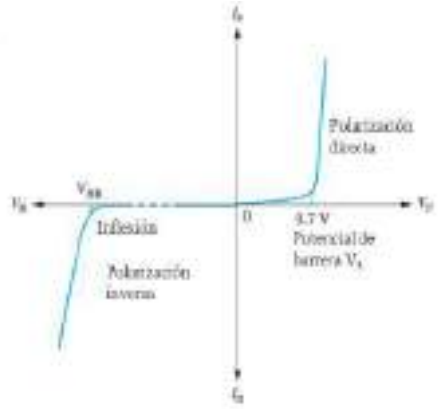
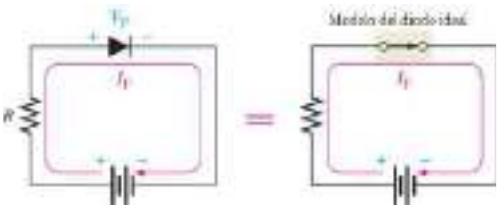


Figura 25: Curva característica V-I completa de un diodo de silicio.

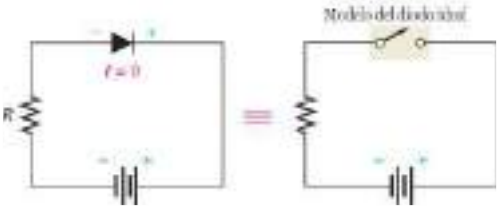
Aproximaciones: Modelos del diodo

Para motivos de estudio y análisis del diodo al momento de diseñar circuitos que los incluyan, no es fácil utilizar las características de los diodos reales y debido a su complejidad de diseño, se utilizan tres aproximaciones: el **modelo ideal**, el **modelo práctico** y el **modelo completo**.

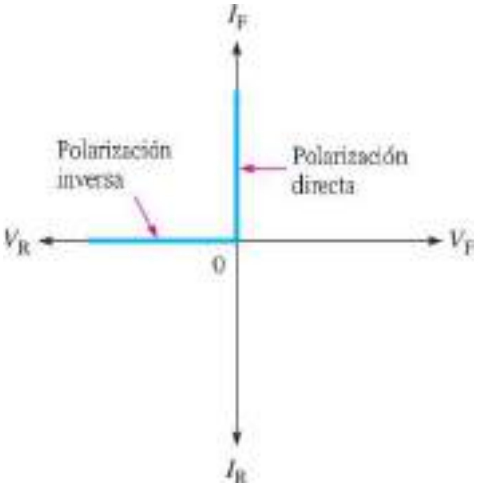
El **modelo ideal** del diodo o simplemente llamado diodo ideal nos presenta un componente totalmente teórico y perfecto, el cual se comporta como un conductor perfecto (resistencia cero) cuando está polarizado en directa y como un aislante perfecto (resistencia infinita) cuando está polarizado en inversa. Es decir, su función sería la de un interruptor que se cierra con polarización directa y se abre con polarización inversa, tal como muestra la Figura 26(a) y Figura 26(b), y la gráfica de V-I sería una 'L' invertida, como se ve en la Figura 26(c). En la práctica es imposible construir un dispositivo con esas características.



(a) Polarización directa



(b) Polarización inversa

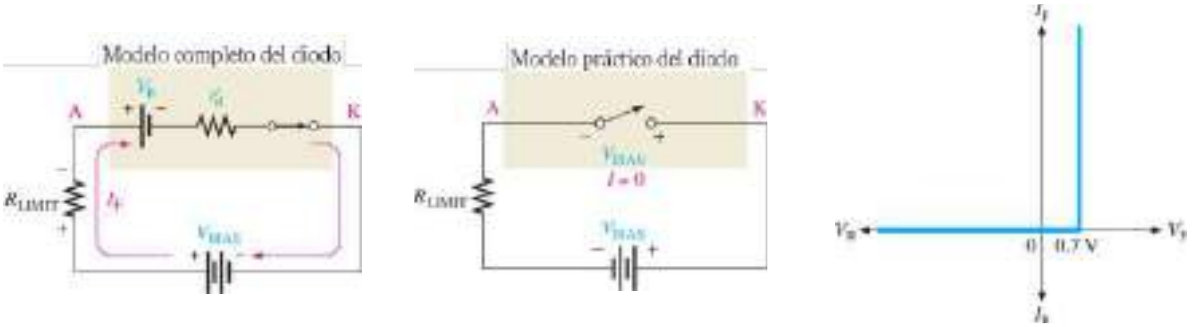


(c) Curva característica V-I del diodo ideal (azul)

Figura 26: El modelo ideal de un diodo

El **modelo práctico** es una aproximación más realista del diodo, porque incluye el potencial de barrera. Cuando el Cuando el diodo está polarizado hacia

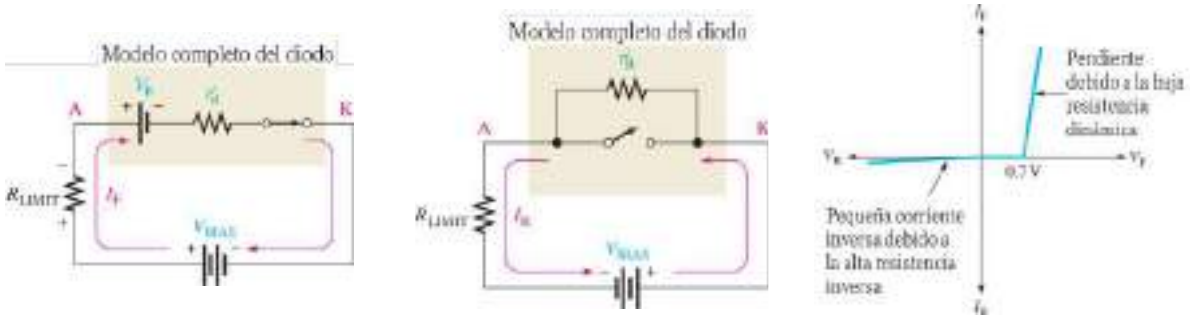
delante, equivale a un interruptor cerrado en serie con una pequeña fuente de tensión equivalente (V_F) igual al potencial de barrera (0,7 V) con el lado positivo hacia el ánodo, como se indica en la Figura 27(a). Esta fuente de tensión equivalente representa el potencial de barrera potencial de barrera que debe ser superado por la tensión de polarización antes de que el diodo conduzca y no es una fuente de tensión activa. Cuando conduce, aparece una caída de tensión de 0,7 V a través del diodo. Cuando el diodo está en polarización inversa, equivale a un interruptor abierto igual que en el modelo ideal como se muestra en la Figura 27(b). El potencial de barrera no afecta a la polarización inversa, por lo que no es un factor. La curva característica del modelo práctico de diodo se muestra en la Figura 27(c).



(a) Polarización directa (b) Polarización inversa (c) Curva característica V-I

Figura 27: El modelo práctico del diodo.

Finalmente tenemos el **modelo completo** de un diodo es la aproximación más precisa e incluye el potencial de barrera, la pequeña resistencia dinámica de polarización directa (r'd) y la gran resistencia interna de polarización inversa (r'R). La resistencia de polarización inversa se tiene en cuenta porque proporciona un camino para la corriente en polarización inversa, la cual se incluye en este modelo de diodo. Cuando el diodo está polarizado directamente, actúa como un interruptor cerrado en serie con la tensión de potencial de barrera equivalente (V_B) y la resistencia pequeña dinámica de polarización directa (r'd), como se indica en la Figura 28(a). Cuando el diodo está en polarización inversamente, actúa como un interruptor abierto en paralelo con la gran resistencia interna de polarización inversa(r'R), como se muestra en la Figura 28(b). El potencial de barrera no afecta a la polarización inversa, por lo que no es un factor. La curva característica del modelo completo de diodo se muestra en la Figura 28(c).



(a) Polarización directa (b) Polarización inversa (c) Curva característica V-I

Figura 28: El modelo completo del diodo.

Características y tipos de diodos

Existen muchas clases de diodos, entre las que tenemos los diodos: Rectificador, Zener, Schottky, ópticos (Diodos Emisores de Luz (*LED – **L**ight **E**mitting **D**iode*), fotodiodos, laser, etc.), entre otros; pero el más común y básico es el diodo rectificador, que permite el flujo de corriente en una sola dirección y que entre sus más frecuentes está el convertir un flujo de corriente alterna (AC) en corriente continua (DC). Por supuesto solo con un solo diodo no es suficiente para este proceso, pero es la base para realizar esta trasformación.

Algunos de estos diodos especiales los veremos en el siguiente tema.

Circuitos de aplicación

Los componentes electrónicos básicos permiten construir circuitos sencillos. Normalmente pueden ser parte de un circuito más grande, como por ejemplo un sistema de control, la placa base de un computador, un televisor, entre otros. El saber identificar un componente pasivo y poder determinar si está en mal estado, puede ser útil en casos de hacer una reparación sencilla, como puede ser el cambiar un condensador o una resistencia dañada.

Como práctica de este primer subtema aprenderemos a utilizar un simulador de circuitos electrónicos, donde crearemos un par de circuitos aplicando lo que hemos visto hasta el momento. El objetivo principal es aprender a utilizar el simulador, para seguir utilizándolo en el resto de la unidad.

El simulador recomendado en esta unidad es el Tinkercad® (www.tinkercad.com). Es un simulador didáctico sencillo y fácil de utilizar. No es exigido utilizar este simulador (existen muchos, incluso de tipo web), pero si es necesario utilizar alguno.

En este caso vamos a crear nuestro primer circuito básico de diodos, para comprobar la polarización directa y la polarización inversa, tal como se muestra el esquemático en la siguiente figura:

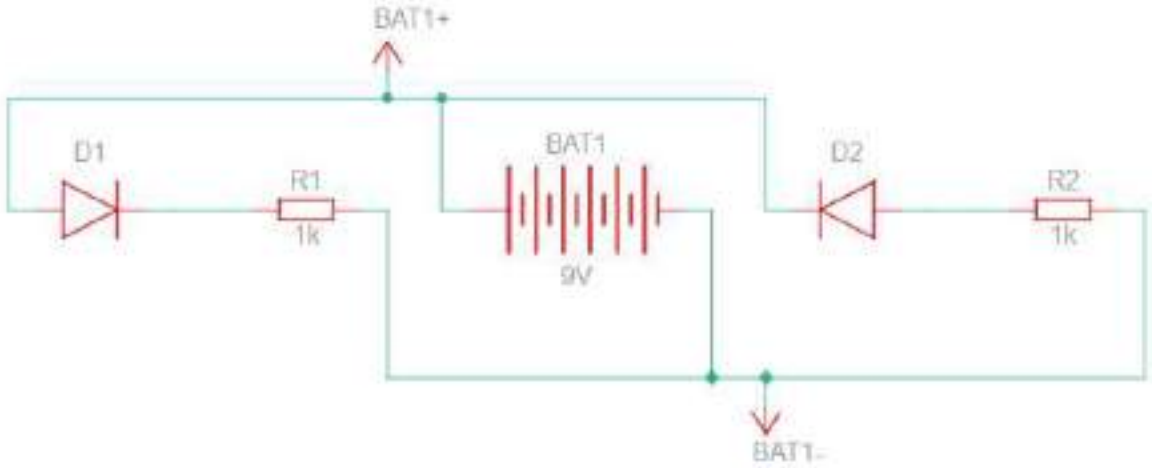


Figura 29: Esquemático de circuito a simular

Para hacer cada circuito vamos a necesitar:

- 1 batería de 9V
- 2 resistencias de 1kΩ
- 2 diodos
- 5 voltímetros

Para interconectar los elementos, usted debe presionar clic izquierdo sobre una de las terminales del componente que desea unir con el componente de destino. Para empezar la simulación, esta se activa al presionar el botón "► Iniciar Simulación" Iniciar simulación. Si usted armó correctamente los circuitos, usted deberá visualizar lo siguiente:

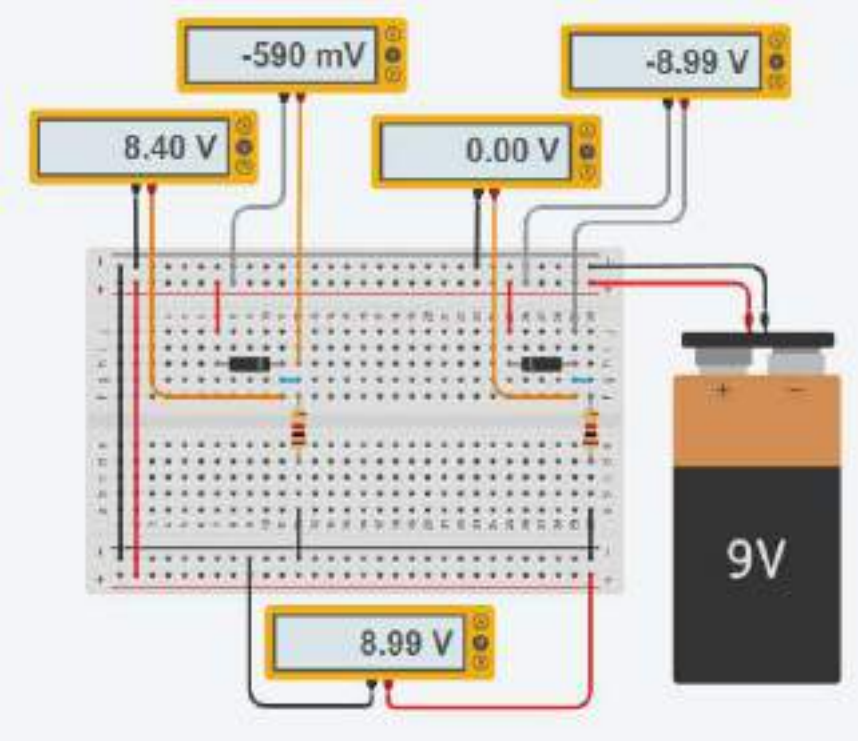


Figura 30: Circuitos simulados

TEMA 2: DISPOSITIVOS SEMICONDUCTORES

Diferentes diodos

Existen varios tipos de diodos, además del diodo rectificador que ya mencionamos. Entre los principales tenemos:

- **Diodo Zener**: A ciertos niveles de voltaje, por lo general bajos, se comporta como un diodo normal. Pero cuando el voltaje que ingresa por el cátodo supera cierto umbral, permite el paso de corriente. Se usa como estabilizador de voltaje.

- **Diodo emisor de luz (*LED – light emitting diode*)**: Son diodos que al permitir el paso de corriente emiten luz. El voltaje directo de estos diodos puede estar entre 1.6 y 4V, y dependiendo del voltaje emiten un color de luz diferente.
- **Fotodiodo**: Pueden ser considerados una celda fotovoltaica (que genera electricidad a partir de la luz) muy diminuta. Cuando un fotodiodo recibe luz, genera una corriente muy pequeña. Son usados en comunicación óptica o en fotometría.
- **Diodo Schottky**: Son diodos que tienen un voltaje directo más bajo que un diodo normal, y genera menos calor. Se los utiliza como rectificadores de baja pérdida de tensión.

Diodo Zener | LED | Fotodiodo | Diodo Schottky

SABÍAS QUE

Los semiconductores pueden generar carga cuando se exponen a la luz, por lo que los diodos por lo general se empacan con materiales opacos para no afectar su funcionamiento, con excepción obviamente de los LED y los fotodiodos.

VIDEO

En este video puedes revisar de nuevo cómo funcionan los diodos
https://www.youtube.com/watch?v=aPY3I8pG478

Diodo Zener

Hasta ahora hemos hablado del diodo tradicional, y de cómo funciona este, pero existen como se ha dicho antes existen otros tipos de diodos, pero solo vamos a hablar del Zener, un diodo Zener es un dispositivo de unión *pn* de silicio que está diseñado para funcionar en la región de ruptura inversa. El voltaje de ruptura de un diodo Zener se establece controlando cuidadosamente el nivel de dopaje durante la fabricación. Recordemos, a partir de la discusión de la curva característica del diodo normal tratado en el Tema 1: y que sucede cuando un diodo alcanza la zona de ruptura inversa, ahora en el diodo Zener su tensión permanece casi constante, aunque la corriente cambie drásticamente. Esta característica voltio-amperio se muestra de nuevo en la Figura **.**-31 con la región de funcionamiento normal de los diodos Zener mostrada como un área sombreada. Su símbolo se muestra en la Figura **¡Error! No hay texto con el estilo especificado en el documento.**-32,

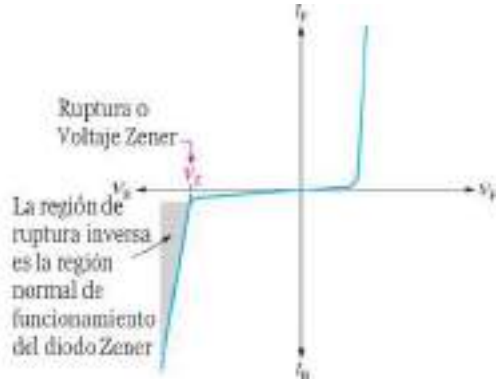


Figura 31: Característica V-I general del diodo Zener.

donde en lugar de una línea recta que representa el cátodo, el diodo Zener tiene una línea doblada que recuerda la letra Z (de Zener).

Ruptura Zener

Los diodos Zener están diseñados para funcionar en ruptura inversa. Los dos tipos de ruptura inversa en un diodo Zener son la avalancha y la Zener. El efecto de avalancha se produce tanto en los diodos rectificadores como en los Zener a una tensión inversa suficientemente alta. La ruptura Zener se produce en un diodo Zener a bajas tensiones inversas. Un diodo Zener está fuertemente dopado para reducir la tensión de ruptura. Esto provoca una región de agotamiento muy fina. Como resultado, existe un intenso campo eléctrico dentro de la región de agotamiento. Cerca del voltaje de ruptura del Zener (V_Z), el campo es lo suficientemente intenso como para extraer electrones de sus bandas de valencia y crear corriente.

Los diodos Zener con voltajes de ruptura inferiores a 5 V aproximadamente funcionan predominantemente en ruptura Zener. Los que tienen tensiones de ruptura superiores a aproximadamente 5 V operan predominantemente en ruptura de avalancha. Sin embargo, ambos tipos se denominan diodos Zener. Los Zener están disponibles en el mercado con tensiones de ruptura de menos de 1 V a más de 250 V con tolerancias especificadas del 1% al 20%.

Figura 32: Símbolo del diodo Zener.

Características de la ruptura

La figura 33 muestra la parte inversa de la curva característica de un diodo Zener. Obsérvese que a medida que se incrementa el voltaje inverso (V_R), la corriente inversa (I_R) se mantiene extremadamente pequeña hasta el "ángulo" de la curva. La corriente inversa también se denomina corriente Zener, I_Z. En este punto, comienza el efecto de ruptura; la resistencia interna del Zener, también llamada impedancia Zener (Z_Z), comienza a disminuir a medida que la corriente inversa aumenta rápidamente. A partir de la parte inferior de la curva, el voltaje de ruptura del Zener (V_Z) permanece esencialmente constante, aunque aumenta ligeramente a medida que aumenta la corriente Zener, I_Z.

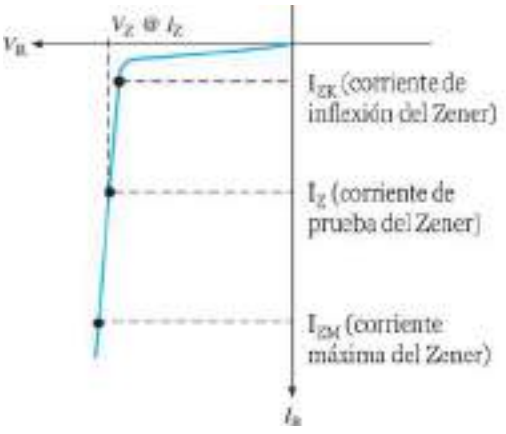


Figura 33: Característica inversa de un diodo Zener de un diodo Zener. VZ se suele especificar a un valor de la corriente Zener conocido como la corriente de prueba.

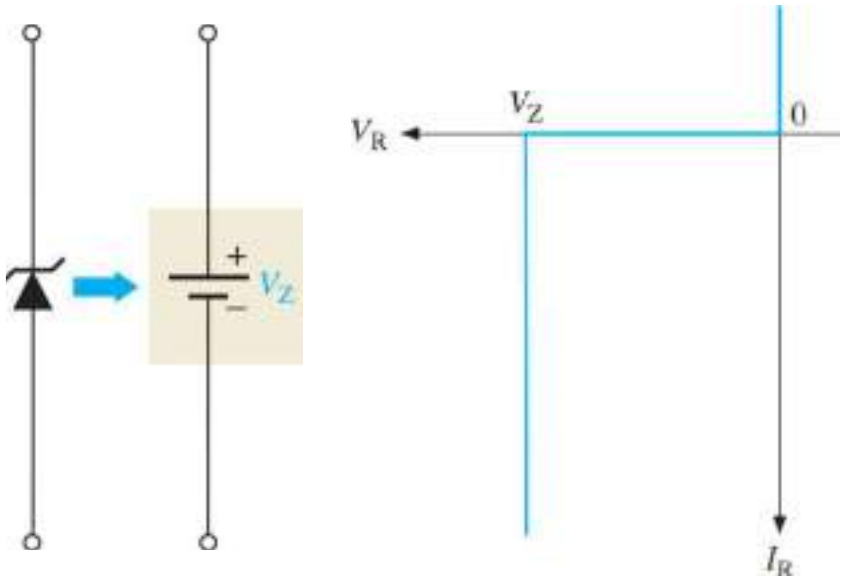
Regulación Zener

La capacidad de mantener el voltaje inverso a través de sus terminales esencialmente constante es la característica clave del diodo Zener. Un diodo Zener

que funciona en ruptura puede actuar como un regulador de voltaje de baja corriente porque mantiene un voltaje casi constante a través de sus terminales en un rango específico de valores de corriente inversa. Es necesario mantener un valor mínimo de corriente inversa, I_{ZK}, para mantener el diodo en ruptura para la regulación del voltaje. En la curva de la Figura**.**-33 se puede ver que cuando la corriente inversa se reduce por debajo del valor de la curva, la tensión disminuye drásticamente y se pierde la regulación. Además, existe una corriente máxima, I_{ZM}, por encima de la cual el diodo puede resultar dañado debido a la excesiva disipación de potencia. Así que, básicamente, el diodo Zener mantiene un voltaje casi constante a través de sus terminales para valores de corriente inversa que van de I_{ZK} a I_{ZM}. El voltaje nominal del Zener, V_Z, suele especificarse en una hoja de datos para un valor de corriente inversa denominado corriente de prueba del Zener.

Circuitos equivalentes Zener: modelos ideal y práctico

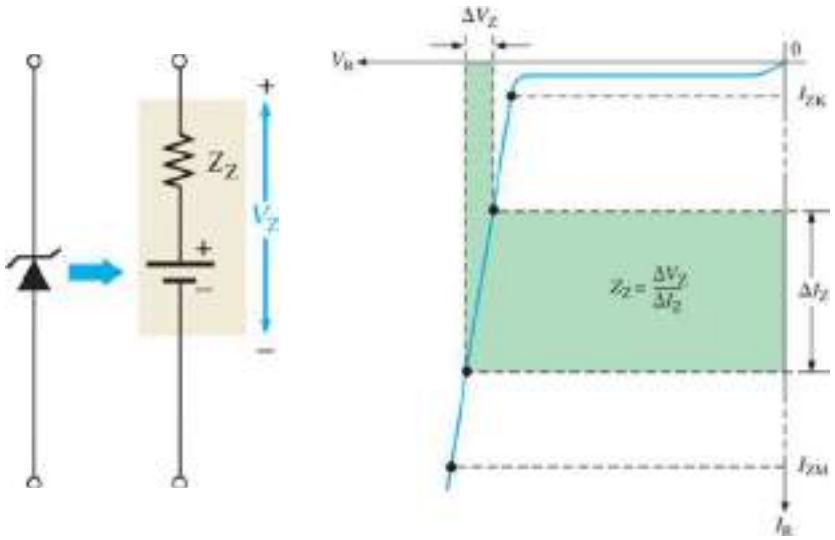
La Figura 34 muestra el **modelo ideal** (primera aproximación) de un diodo Zener en ruptura inversa y su curva característica ideal. Tiene una caída de tensión constante igual a la tensión nominal del Zener. Esta caída de tensión constante a través del diodo Zener producida por la descomposición inversa se representa con un símbolo de tensión continua, **aunque el diodo Zener no produce una tensión**.



(a) Modelo ideal (b) Curva característica del modelo ideal

Figura 34: Circuito equivalente del diodo Zener ideal y la curva característica.

La Figura 35(a) representa el **modelo práctico** (segunda aproximación) de un diodo Zener, donde se incluye la impedancia (resistencia) Zener, Z_Z. Como la curva de voltaje real no es idealmente vertical, como en el modelo anterior, un cambio en la corriente Zener (Δ_{IZ}) produce un pequeño cambio en el voltaje Zener (Δ_{VZ}), como se ilustra en la Figura 35(b). Por la ley de Ohm, la relación entre Δ_{VZ} y Δ_{IZ} es la impedancia, como se expresa en la siguiente ecuación: $Z_Z = \Delta V_Z/\Delta I_Z$.



(a) Modelo práctico

(b) Curva característica.
La pendiente está exagerada para ilustrar

Figura 35: Circuito equivalente del diodo Zener
práctico circuito equivalente y la curva característica que ilustra Zz

Dado que Z_Z se define como un cambio de tensión sobre un cambio de corriente, es una resistencia dinámica. Normalmente, Z_Z se especifica con la corriente de prueba del Zener. En la mayoría de los casos, se puede asumir que Z_Z es una pequeña constante en todo el rango de valores de corriente del Zener y es puramente resistiva. Es mejor evitar el funcionamiento de un diodo Zener cerca de la curva porque la impedancia cambia drásticamente en esa zona. impedancia cambia drásticamente en esa zona.

Circuitos de aplicación de diodos

Los diodos son ampliamente utilizados en dispositivos electrónicos, principalmente para convertir la corriente alterna de la red eléctrica en corriente continua, la cual es utilizada en electrónica. Veremos algunos circuitos donde se aplica este componente. Estos circuitos pueden implementarse en el simulador TinkerCAD

Rectificador de media onda

En la Figura 36 se muestra un circuito rectificador de media onda. La fuente de alterna produce una tensión sinusoidal (corriente alterna). Para el circuito de la derecha Figura 36(a), durante el ciclo positivo de la onda, el diodo está polarizado directamente, mientras que para el ciclo negativo el diodo está polarizado inversamente, permitiendo visualizar en el osciloscopio únicamente el ciclo positivo de la onda, mientras que para el circuito de la izquierda Figura -36(b), durante el ciclo positivo de la onda, el diodo está polarizado inversamente, mientras que para el ciclo negativo el diodo está polarizado directamente, permitiendo visualizar en el osciloscopio únicamente el ciclo negativo de la onda.

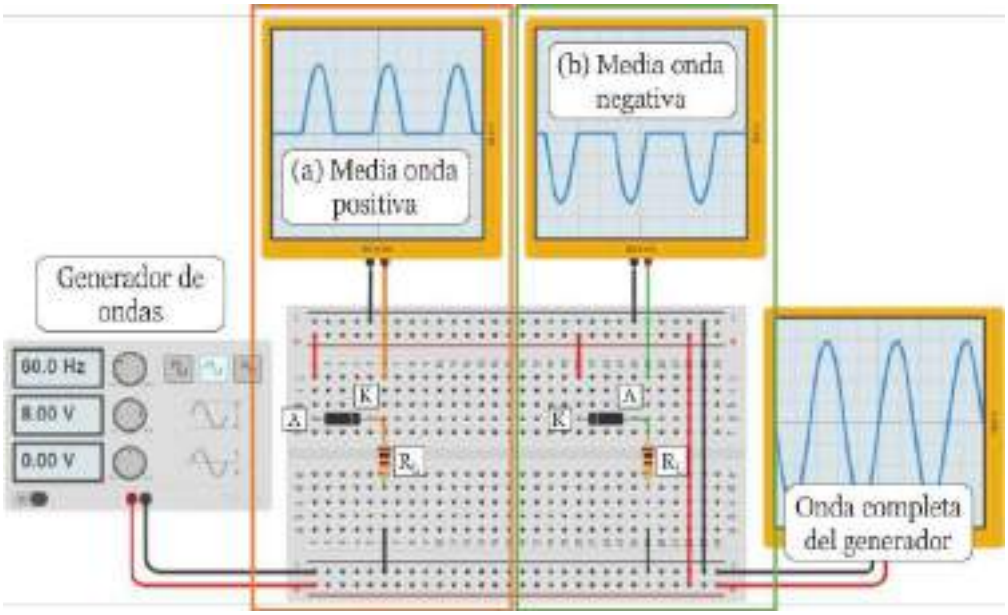


Figura 36: Rectificador de media onda: (a) ciclo positivo, (b) ciclo negativo

Como se ha explicado en este compendio, para el caso del circuito de la Figura 36(a), el diodo solo permite el paso de la corriente durante el ciclo positivo de la señal, tal como se muestra en su respectivo osciloscopio, y en el caso del circuito de la Figura 36(b) el diodo solo permite el paso de la corriente durante el ciclo negativo de la señal, tal como se muestra en su respectivo osciloscopio. Por lo tanto, al dejar pasar uno de los ciclos de la señal, a la forma de como resultante se le denomina señal de media onda. Estas media ondas se las puede comparar con la onda completa mostrada por el tercer osciloscopio.

Una señal de media onda como la mostrada es una tensión continua pulsante que crece hasta un máximo, decrece hasta cero y permanece así durante el semiciclo negativo. Éste no es el tipo de tensión continua que se necesita para los equipos electrónicos. Lo que se necesita es una tensión constante.

En circuito ejemplo de la Figura 36, se conecta a una fuente de $4V_P$ (voltaje *peak*[1]), es decir $8V_{PP}$ (voltaje *peak-to-peak*) y 60Hz y utiliza tres osciloscopios para ver la forma de onda para cada salida (señal de entrada, rectificador de media onda positiva y rectificador de media onda negativa).

COMPRUEBA TU APRENDIZAJE

Si usted es observador, notará que en las medias ondas resultantes en los osciloscopios de la Figura 36, su valor peak es más pequeño que el de la onda de entrada.
¿Podría usted explicar el motivo de por qué estas señales resultantes son más pequeñas que la señal de entrada?

1. La palabra peak, es utilizado para referirse a la parte más alta o baja de la onda.

Rectificador de onda completa

Para este tema, pueden existir dos circuitos que rectifican la onda de forma completa. El primero que vamos a ver, utiliza un puente de diodos (4 diodos conectados, ver Figura 37) interconectados, teniendo así dos entradas y dos salidas, donde las entradas reciben la señal AC y las salidas entregan la señal DC (onda completamente rectificada).

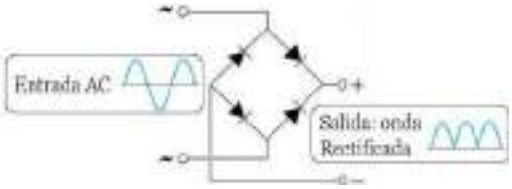


Figura 37: Puente de diodos con sus entradas y salidas.

En la Figura 38, se puede ver la implementación de un rectificador de onda completa a través del puente de diodos, aunque en este caso se está utilizando un generador de ondas para generar la onda seno aplicada a la entra AC del puente de diodos.

DATOS ÚTILES

En el circuito implementado se utiliza un generador de ondas para tener la señal de entrada, pero en la practica la señal de onda proviene de la red eléctrica que en el Ecuador es de 120Vrms (rms o voltaje eficaz) a 60Hz, en otros países el estándar es 220V a 50Hz o 60 Hz, por lo tanto, es necesario utilizar un transformador eléctrico para bajar ese voltaje de la red eléctrica.

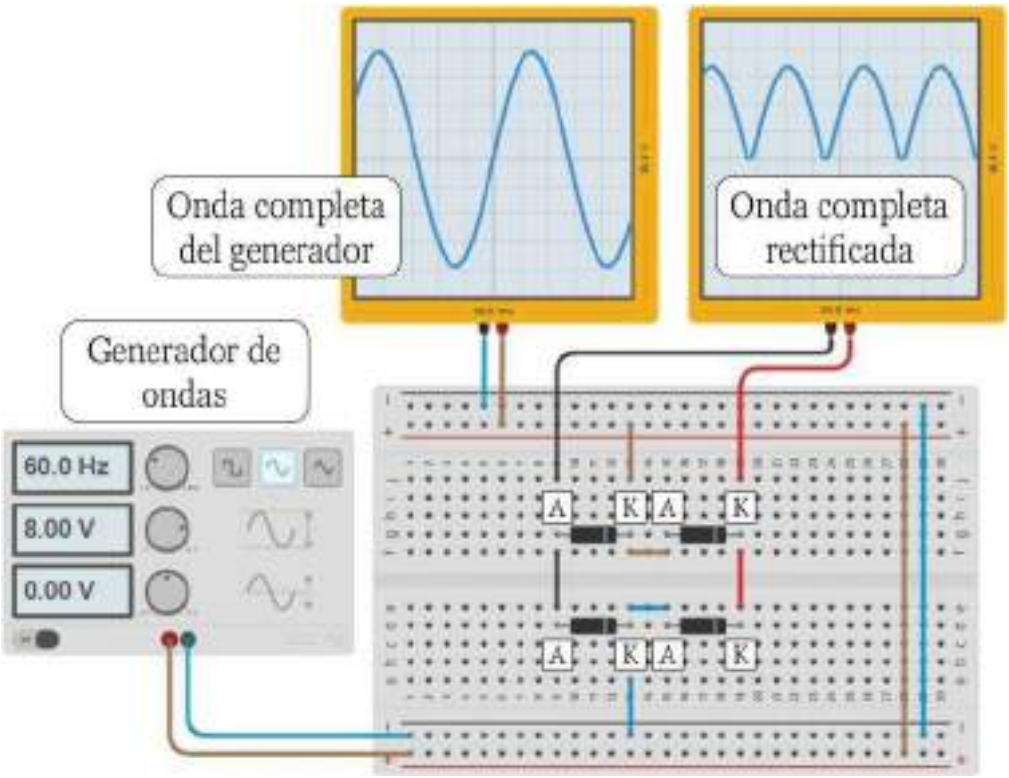


Figura 38: Circuito rectificador de onda completa utilizando un puente de diodos y su onda de salida

DATOS ÚTILES

El **transformador** es una máquina eléctrica utilizada para bajar el voltaje de la red eléctrica a voltajes adecuados para los equipos electrónicos. El transformador hace este cambio de voltaje con la ayuda de dos embobinados llamados Np y Ns, donde Np es el número de vueltas de la bobina primaria y Ns es el número de vueltas de la bobina secundaria, además son utilizados para identificar el tipo de transformador y como se relacionan las bobinas, ej.:

- Si Np es igual a 4 y Ns es igual a 1, la relación se lee 4:1, esto quiere decir que, por cada 4 vueltas en la bobina primaria, la bobina secundaria tendría 1 vuelta, además sirve para saber la relación de los voltajes de entrada y salida, porque si en la entrada del NP tenemos 120Vrms, para esta relación a la salida del NS tendremos 30Vrms.

El transformador eléctrico para construir una fuente de poder de corriente continua, siempre ser instalado antes del puente de diodos.

El otro circuito rectificador de onda completa a diferencia del circuito anterior utiliza forzadamente un transformador con una bobina en el primario y 2 bobinas conectadas en el secundario, la unión de las dos bobinas se llama toma central (*center tap*) como el de la Figura 39 y únicamente dos diodos rectificadores.

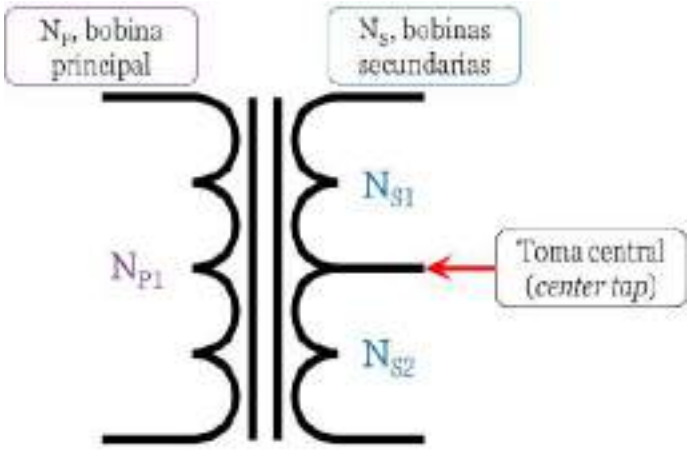


Figura 39: Símbolo de un transformador con dos bobinas secundarias N_{S1} y N_{S2}, y una toma central (center tap)**.**

Para el caso de la simulación, el TinkerCAD no cuenta con transformadores eléctricos, por lo cual para simular las salidas de los embobinados secundarios se conectaron dos generadores de onda en serie. Entonces, observando la implementación en la Figura 40, la señal sinusoidal del generador #1 (aquí equivale a la salida del embobinado N_{S1}) es rectificada por el diodo # 1, mientras que la señal sinusoidal del generador #2 (aquí equivale a la salida del embobinado N_{S2}) es rectificada por el diodo # 2. Por lo tanto, la salida del diodo #1 que es igual a la salida de la Figura 36(a) y la salida del diodo #2 es igual a la salida de la Figura 36(b). Pero al unirse los cátodos de los diodos #1 y #2, se produce una suma de señales, produciendo así la rectificación de la onda completa a la salida de los diodos, igual a la observable en la Figura 40.

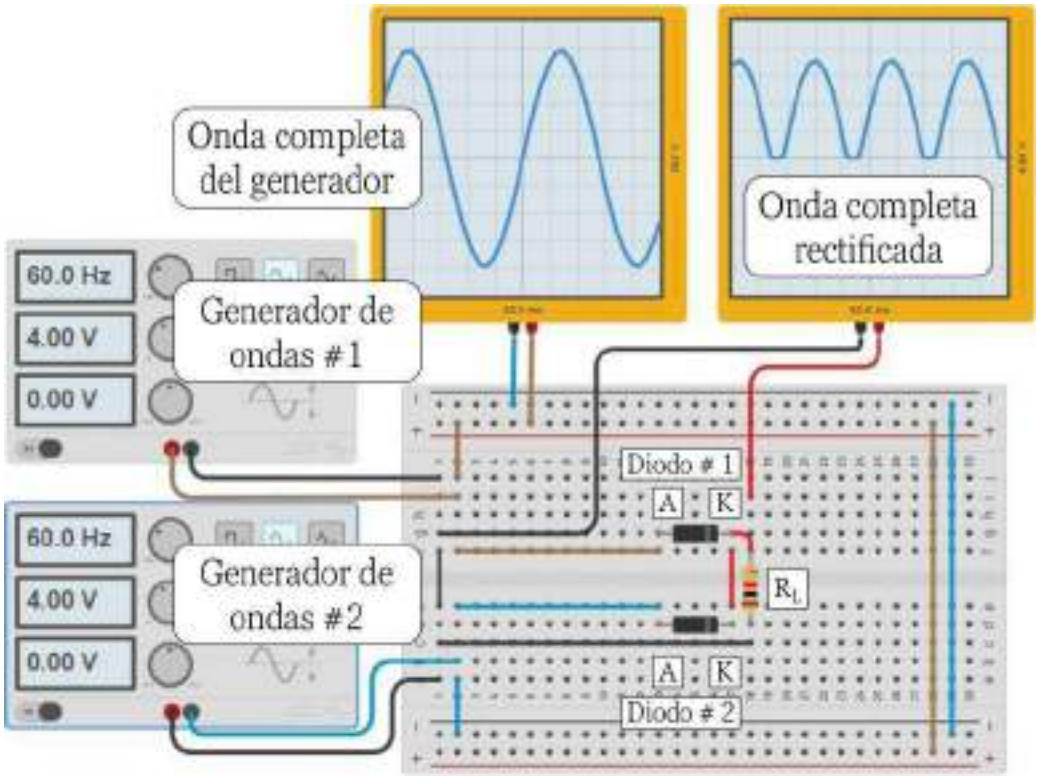


Figura 40: Circuito rectificador de onda completa utilizando un transformador con toma central y dos diodos, además de su onda de salida

VIDEO

En este video puedes revisar de nuevo cómo funcionan algunos de los circuitos rectificadores https://www.youtube.com/watch?v=Fk4Z8a2Xszk

TEMA 3: TRANSISTORES

Funcionamiento de los Transistores

Podríamos decir que el transistor sería el primer dispositivo electrónico como tal, por estar construido con 3 o más *junturas pn*, aunque el diodo también es considerado un dispositivo electrónico por ser la base de la electrónica moderna, pero gracias al descubrimiento del transistor se podría decir que nació la electrónica moderna y la era digital, ya que hasta antes de 1947 la electrónica utilizaba tubos al vacío, haciendo que estos, que los equipos de la época sean muy grandes, pero con el descubrimiento del transistor aparecieron los circuitos integrados en un solo chip, de hecho, todos los circuitos integrados actualmente están fabricados a base de transistores.

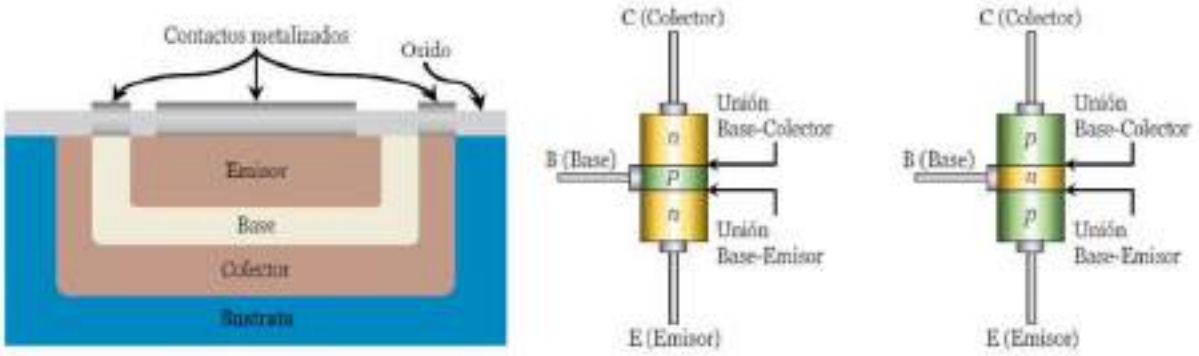
El germanio y el silicio son los principales materiales utilizados para crear transistores, con impurezas introducidas para determinar el tipo de conductividad (el tipo *n* tiene un exceso de electrones libres; el tipo *p*, una deficiencia., tal como hemos visto antes). La conducción se realiza por medio de electrones libres y

huecos (Graf, 1999). Entonces, un transistor es un dispositivo semiconductor activo que al tener 3 o más uniones *pn*, tiene tres o más electrodos y es capaz de realizar varias funciones, incluidas: interruptor, amplificador, etc. Entre los transistores más usados tenemos: transistores de unión bipolar (*BJT - Bipolar Junction Transistors*) y transistores de efecto de campo (*FET - Field-Effect Transistors*). Los BJT tienen 3 pines: base (B), colector (C) y emisor (E), el FET por su parte dependiendo de si es un JFET o un MOSFET, puede tener 3 o 4 pines: puerta (G - *Gate*), drenador (D - *Drain*) y fuente (S - *Source*) y en el caso del MOSFET el cuerpo (B - *Body*)

Los transistores pueden venir en varias presentaciones, ya sea con terminales largos para su montaje, encapsulados, o de montaje superficial

Transistores bipolares (BJT): funcionamiento y características.

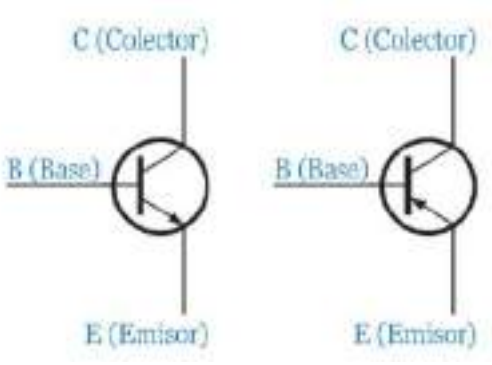
El BJT se construye con tres regiones semiconductoras dopadas separadas por dos uniones *pn*, como se muestra en la estructura planar epitaxial de la Figura 41(a). Las tres regiones se denominan emisor, base y colector. Las representaciones físicas de los dos tipos de BJT se muestran en la Figura 41(b) y (c). Un tipo consta de dos regiones *n* separadas por una región *p* (*npn*), y el otro tipo consta de dos regiones *p* separadas por una región *n* (*pnp*). El término bipolar se refiere al uso de huecos y electrones como portadores de corriente en la estructura del transistor.



(a) Estructura plana epitaxial básica (b) *npn* (c) *pnp*

Figura 41: Construcción básica de un BJT.

La unión *pn* que une la región base y la región del emisor se denomina unión base-emisor (BE). La unión *pn* que une la región de la base y la región del colector se denomina unión base-colector (BC), como se indica en la Figura 41(b) y (c). Un conector une a cada una de las tres regiones. Estos conectores se denominan E, B y C para el Emisor, la Base y el Colector, respectivamente. La región de la base está ligeramente dopada y es muy delgada en comparación con el emisor fuertemente dopado y el colector moderadamente dopado, y la región del colector moderadamente dopada. Debido a esta diferencia en los niveles de dopaje niveles de dopaje, el emisor y el colector no son intercambiables. La Figura 42 muestra los símbolos esquemáticos de los transistores de unión bipolar *npn* y *pnp*.



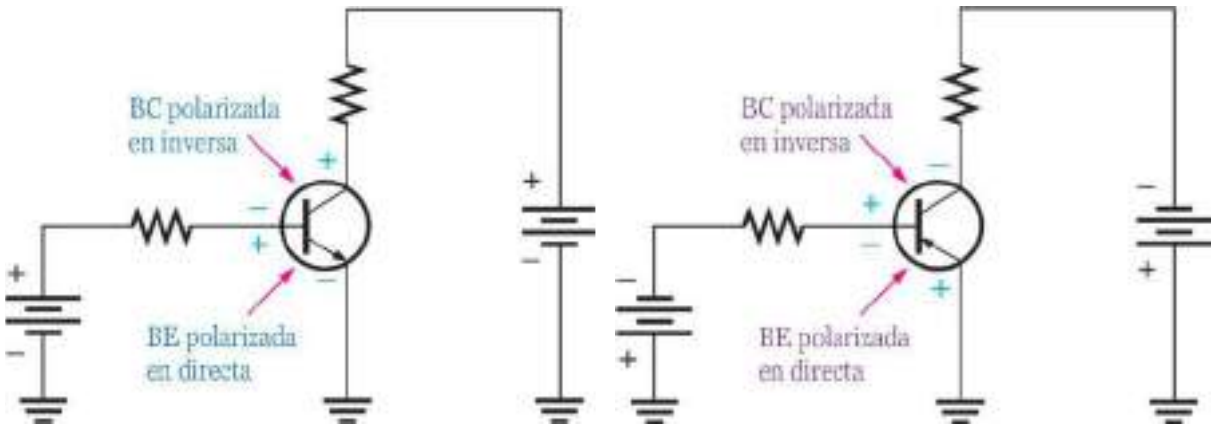
(a) *npn* (b) *pnp*

Figura 42: Simbología del BJT

El BJT tiene dos polaridades (de ahí su nombre), y puede usarse como amplificador y como conmutador. En las siguientes figuras puede verse la estructura interna de un transistor BJT. La región inferior es el emisor, la región intermedia es la base y la región superior es el colector. En un transistor real, la región de la base es mucho más estrecha.

Puesta a punto (Biasing)

La Figura 43 muestra una disposición de polarización para los BJTs *npn* y *pnp* para su funcionamiento como amplificador. Observe que en ambos casos la unión base-emisor (BE) está polarizada hacia delante y la unión base-colector (BC) tiene polarización inversa. Esta condición se denomina *polarización en directa-inversa*.



(a) *npn* (b) *pnp*

Figura 43: Polarización en directa-inversa

Operación

Para entender el funcionamiento de un transistor, examinemos lo que ocurre dentro de la estructura *npn*. La región emisora de tipo *n*, fuertemente dopada, tiene una densidad muy alta de electrones de la banda de conducción (libres) como se

indica en la Figura 44. Estos electrones libres se difunden fácilmente a través de la unión BE de tipo frontal hacia la región base de tipo *p*, ligeramente dopada y muy delgada, como indica la flecha ancha. La base tiene una baja densidad de huecos, que son los portadores mayoritarios, como representan los círculos blancos. Un pequeño porcentaje del número total de electrones libres inyectados en la región de la base se recombina con los huecos y se mueve como electrones de valencia a través de la región de la base y hacia la región emisora como corriente de huecos, indicada por las flechas rojas.

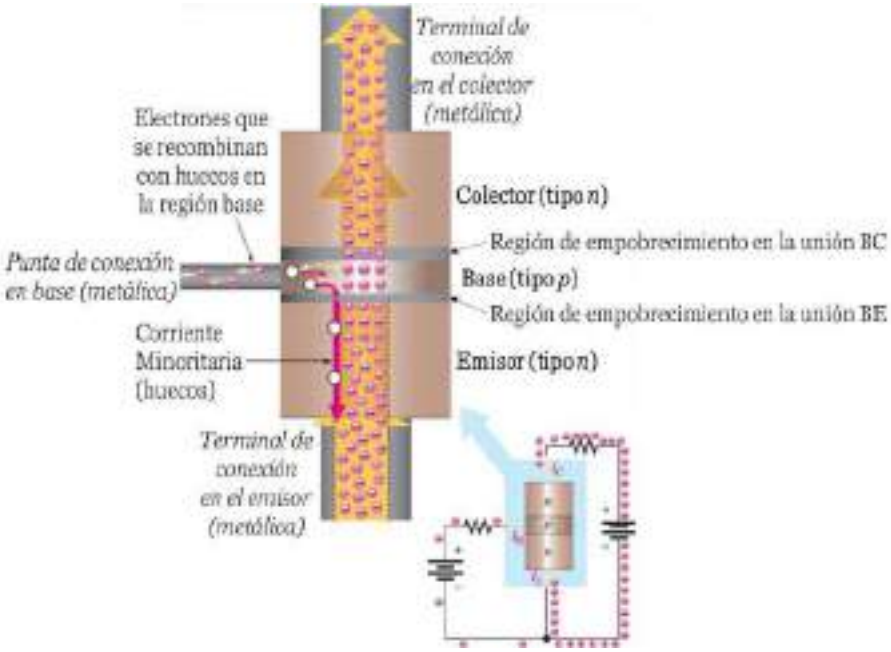


Figura 44: Operación de un BJT que muestra el flujo de electrones

Cuando los electrones que se han recombinado con los huecos como electrones de valencia abandonan la estructura cristalina de la base, se convierten en electrones libres en el conductor de la base metálica y producen la corriente de la base externa. La mayoría de los electrones libres que han entrado en la base no recombinan con agujeros porque la base es muy fina. A medida que los electrones libres se mueven hacia la de polarización inversa, son arrastrados a la región del colector por la atracción del voltaje positivo de la alimentación del colector. Los electrones libres se mueven a través de la región del colector, hacia el circuito externo, y luego regresan a la región del emisor junto con la corriente de la base, como se indica. La corriente de emisor es ligeramente mayor que la de colector debido a la pequeña corriente de base que se separa de la corriente total inyectada en la región de base desde el emisor.

SABÍAS QUE

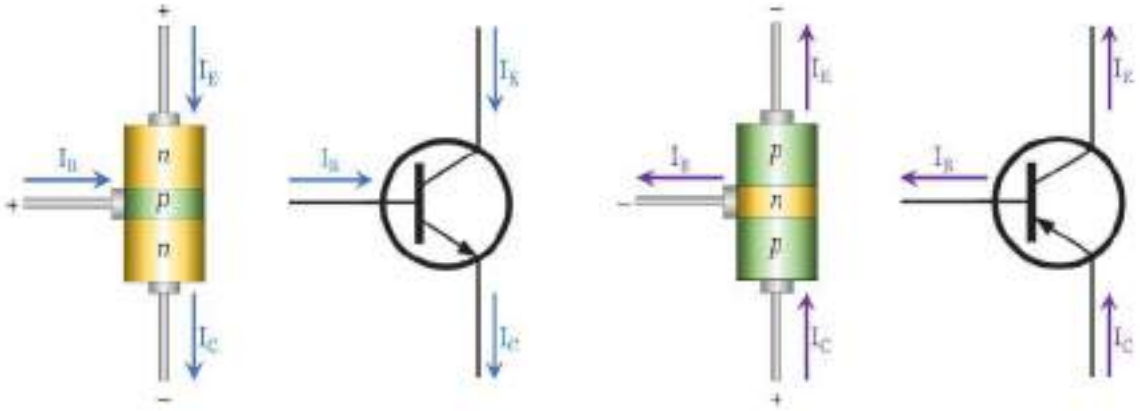
Se llama "portadores mayoritarios" a los electrones libres en un material de tipo n, y a los huecos en un material de tipo p.

Corrientes del transistor

Las direcciones de las corrientes en un transistor *npn* y su símbolo esquemático son las que se muestran en la Figura 4-5(a); las de un transistor *pnp* se muestran en la Figura 4-5(b). Observe que la flecha del emisor dentro de los símbolos del transistor apunta en la dirección de la corriente convencional. Estos diagramas muestran que la corriente de emisor (IE) es la suma de la corriente de colector (IC) y la corriente de base (IB), expresada como sigue:

$$I_E = I_B + I_C$$

Como ya se ha dicho, IB es muy pequeña en comparación con IE o IC. Los subíndices en mayúsculas indican valores de corriente continua.



(a) *npn* (b) *pnp*

Figura 45: Corrientes del transistor

Características y curvas

Beta DC (β_{DC}) y Alfa DC (α_{DC})

La ganancia de corriente de DC de un transistor es la relación entre la corriente DC de colector (I_C) y la corriente de DC base (I_B) y se denomina beta de DC (β_{DC}).

$$\beta_{DC} = \frac{I_C}{I_B}$$

Los valores típicos de β_{DC} oscilan entre menos de 20 y 200 o más. β_{DC} suele designarse como un parámetro híbrido equivalente (h), h_{FE}, en las hojas de datos de los transistores. Por ahora, todo lo que necesita saber es que $h_{FE} = \beta_{DC.}$

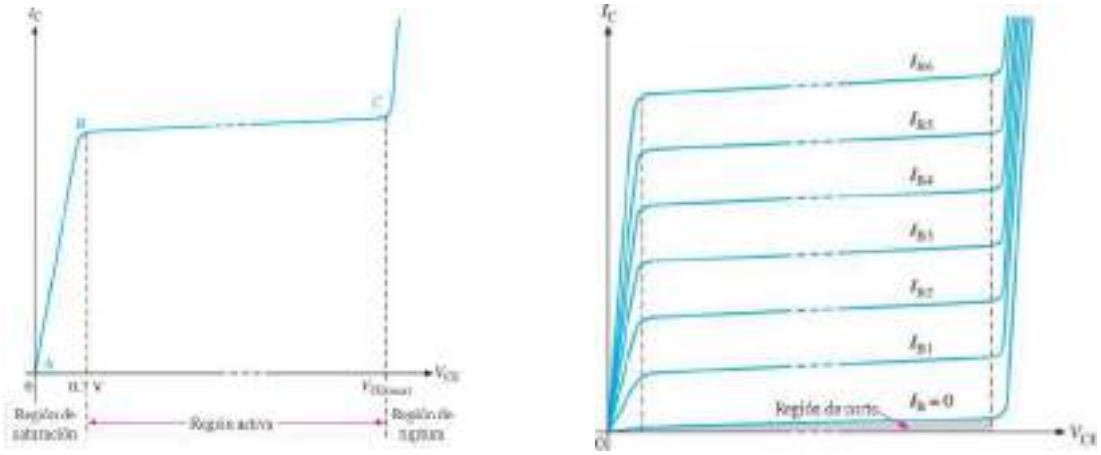
La relación entre la corriente DC del colector (I_C) y la corriente DC del emisor (I_E) es alfa DC (α_{DC}). El alfa es un parámetro menos utilizado que beta en los circuitos de transistores.

$$\alpha_{DC} = \frac{I_C}{I_E}$$

En general, los valores de α_{DC} van desde 0.95 hasta 0.99 o más, aunque α_{DC} siempre es menor que 1. La razón es que I_C siempre es un poco menor que I_E en una cantidad de I_B. Por ejemplo, si I_E = 100 mA e I_B = 1 mA, entonces I_C = 99 mA y α_{DC} = 0.99.

Un transistor tiene cuatro regiones de operación distintas: activa, corte, saturación y disrupción. Cuando se emplean transistores para amplificar señales débiles, estos trabajan en la región activa. Algunas veces, la región activa se denomina región lineal, porque las variaciones en la señal de entrada producen variaciones proporcionales en la señal de salida. Las regiones de saturación y de corte resultan útiles en los circuitos digitales y de computadoras, y se conocen como circuitos de conmutación.

La figura a continuación muestra como varía el funcionamiento del transistor en las diferentes regiones. La primera de ellas es la región intermedia donde V_{CE} toma valores entre 1 y 40 V, y tiene lugar el funcionamiento normal del transistor. Los cambios en la tensión del colector no tienen efecto en la corriente de mismo. Esta región es la región activa. Gráficamente, la región activa es la parte casi horizontal de la curva. En la figura del lado derecho se muestran también un conjunto de curvas del colector, en las que la corriente en el colector puede variar, dependiendo de la corriente de base.



(a) Curva de IC contra VCE para un valor de I_B

(b) Familias de curvas I_C contra V_{CE} para varios valores de I_B ($I_{B1} < I_{B2} < I_{B3}$, etc.)

Figura 46: Curvas características de colector.

Circuitos con transistores BJT

Usando el simulador TinkerCAD, realizaremos un circuito que utiliza 2 fuentes de poder diferentes, una de $9V_{DC}$ y otra de $18V_{DC}$, un motor DC, un transistor tipo

npn (2n2222), 2 resistencias (R_C = 1Ω y R_B = 4.7kΩ), y un pulsador. El transistor está configurado en corte y saturación, entonces al no existir corriente de base (I_B) el motor está apagado, pero al presionar el pulsador, aparece I_B y satura el transistor haciendo que este conduzca la corriente y encienda el motor.

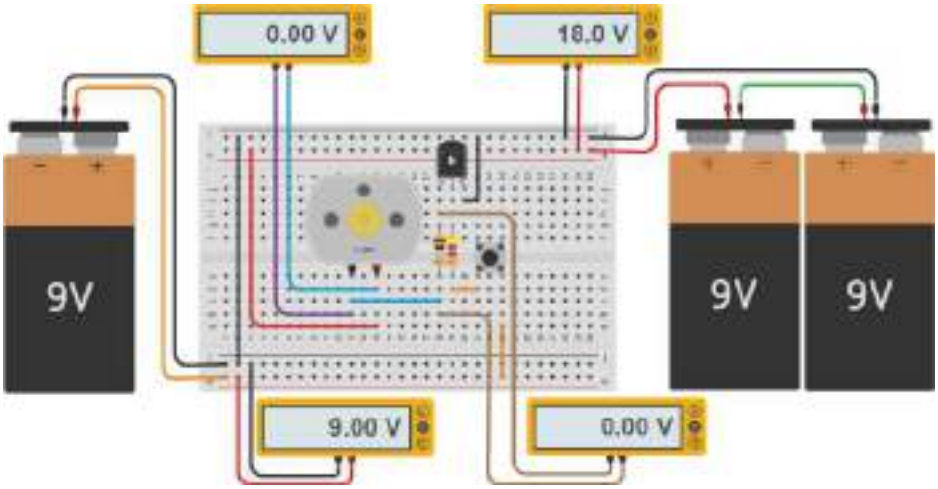


Figura 47: Circuito con el motor apagado

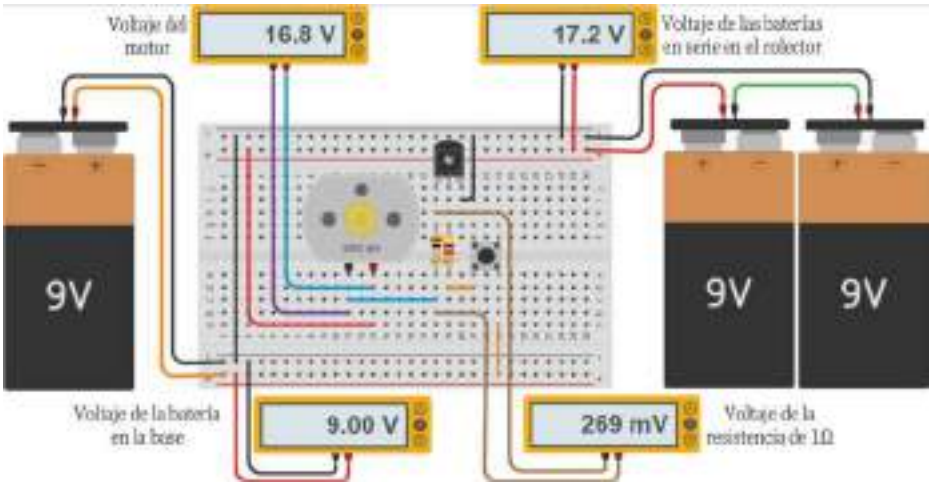


Figura 48: Circuito con el motor encendido

Otra aplicación es el amplificador, para un circuito amplificador, se puede trabajar con polarización de base o polarización de emisor. Un amplificador con polarización de base no es útil para la fabricación en serie, tiene valor didáctico porque se puede utilizar para construir amplificadores más complejos. Un amplificador con polarización de emisor es más estable, y por lo tanto más utilizado.

Los amplificadores pueden utilizarse ya sea para amplificar la tensión o la potencia de una señal eléctrica. Dada la diversidad de aplicaciones, y a que no es el objetivo de esta asignatura profundizar demasiado en este tema, no analizaremos todos los circuitos posibles donde se aplicar transistores bipolares

VIDEO

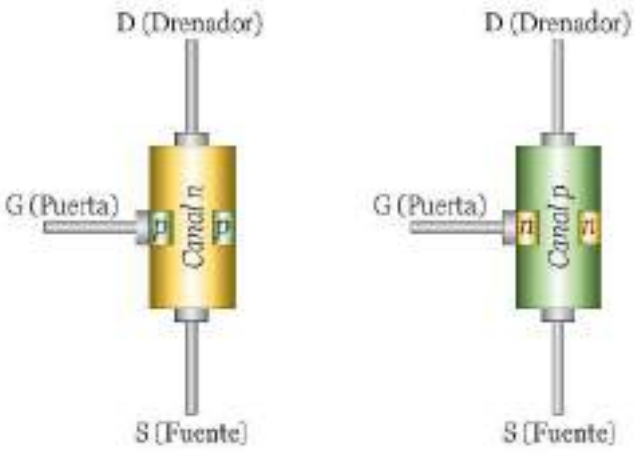
En este video puedes revisar de nuevo cómo funcionan los transistores bipolares, asi como ejemplos de circuitos con este tipo de transistores:
https://www.youtube.com/watch?v=zh7PeHAZRLY

Transistores de efecto de campo (JFET y MOSFET): funcionamiento y características

El transistor de efecto de campo (FET - Field-Effect Transistor) es un tipo de transistor unipolar porque su operación sólo depende de un tipo de carga, electrones libres o huecos. En otras palabras, un FET tiene portadores mayoritarios, pero no portadores minoritarios. Existen dos clases de transistores unipolares: el JFET (Junction Field-Effect Transistor) y el MOSFET (Metal-Oxide semiconductor FET). Veremos cómo funciona cada uno de ellos cada uno de ellos.

JFET

Para fabricar un JFET, el fabricante difunde dos áreas de semiconductor de tipo p en el semiconductor de tipo n, como se muestra en la figura a continuación. Estas regiones p están conectadas internamente para conseguir un sólo terminal externo de puerta.



(a) Canal *n* (b) Canal *p*

Figura 49: Una representación de la estructura básica de los dos tipos de JFET

Para ilustrar el funcionamiento de un JFET, la Figura 50 muestra las tensiones de polarización de corriente continua aplicadas a un dispositivo de canal n. V_{DD} proporciona una tensión de drenaje a fuente y suministra corriente de drenaje a fuente. V_{GG} establece la tensión de polarización inversa entre la puerta y la fuente, como se muestra. El JFET siempre funciona con la unión *pn* puerta-fuente en polarización inversa. La polarización inversa de la unión puerta-fuente produce una región de agotamiento a lo largo de la unión *pn*, que se extiende dentro del canal y por lo tanto aumenta su resistencia al restringir el ancho del canal.

La anchura del canal y, por tanto, la resistencia del canal puede controlarse variando el voltaje de puerta, controlando así la cantidad de corriente de drenaje, I_D.

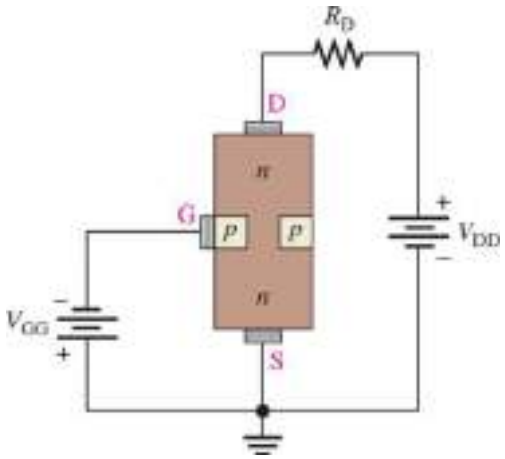
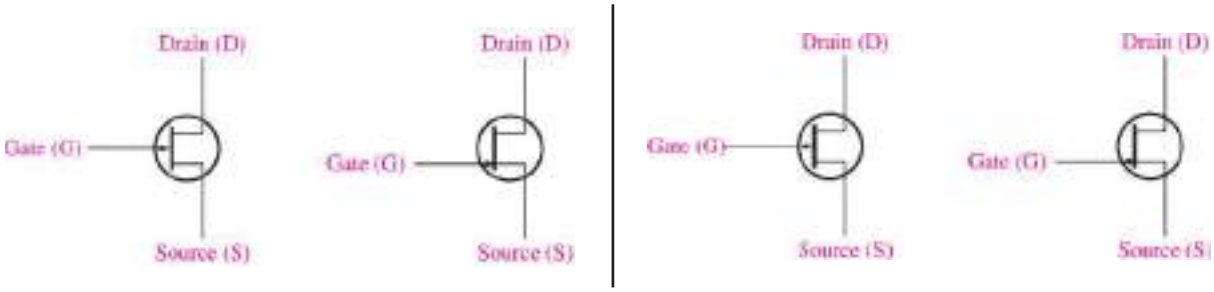


Figura 50: Polarización normal de un JFET de canal *n*

El JFET es un dispositivo controlado por tensión porque una tensión de entrada controla una corriente de salida. En un JFET, la tensión puerta-fuente V_{GS} determina la cantidad de corriente que fluye entre la fuente y el drenador. Si V_{GS} es cero, a corriente máxima de drenador circula a través del JFET; pero si es lo suficiente negativa, las zonas de deplexión se tocarán y la corriente de drenador se cortará.

En las siguientes figuras se muestran el símbolo esquemático del JFET, el de la izquierda es un JFET de canal n, mientras que el de la derecha es un JFET de canal *p*. El funcionamiento de un JFET de canal *p* es complementario, es decir, todas las tensiones y corrientes están invertidas. Para polarizar en inversa un JFET de canal *p*, la puerta tiene que ser positiva respecto a la fuente. Por tanto, V_{GS} se hace positiva.



JFET de canal *n* JFET de canal *p*

Figura 51: simbología esquemática del JFET normal y alterno

Un JFET tiene varias regiones o zonas de funcionamiento. Al igual que los BJT, están las zonas de corte, activa, y de saturación. Pero también se encuentra

la zona óhmica, que es la parte casi vertical de la curva de drenador por debajo del punto de estrangulamiento (Malvino & Bates, 2007).

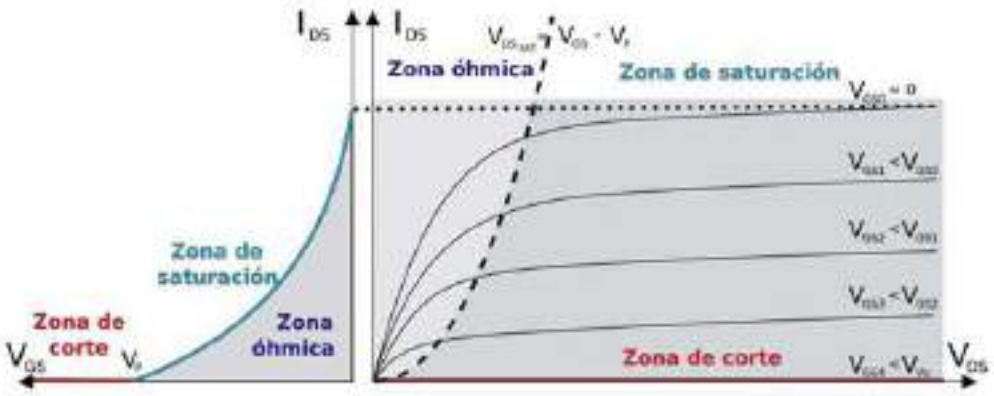


Figura 52: Curvas características de un JFET.

MOSFET

El MOSFET tiene una fuente, una puerta y un drenador. Sin embargo, el MOSFET se diferencia del JFET en que la puerta está aislada del canal. Existen dos clases de MOSFET, el tipo que opera en modo de vaciamiento (D-MOSFET, depletion-mode MOSFET) y el tipo que opera en modo de enriquecimiento (E-MOSFET ,enhancement-mode MOSFET).

La figura siguiente muestra un D-MOSFET, un fragmento de material n con una puerta aislada a la izquierda y una región p a la derecha. La región p se denomina sustrato. Los electrones que fluyen desde la fuente hacia el drenador deben atravesar el estrecho canal existente entre la puerta y el sustrato p. En la parte izquierda del canal hay depositada una delgada capa de dióxido de silicio (SiO2), que es un aislante; mientras que la puerta es de metal.

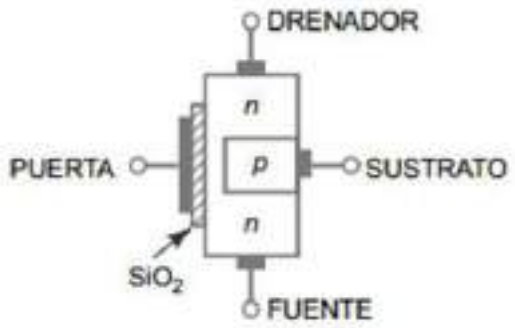
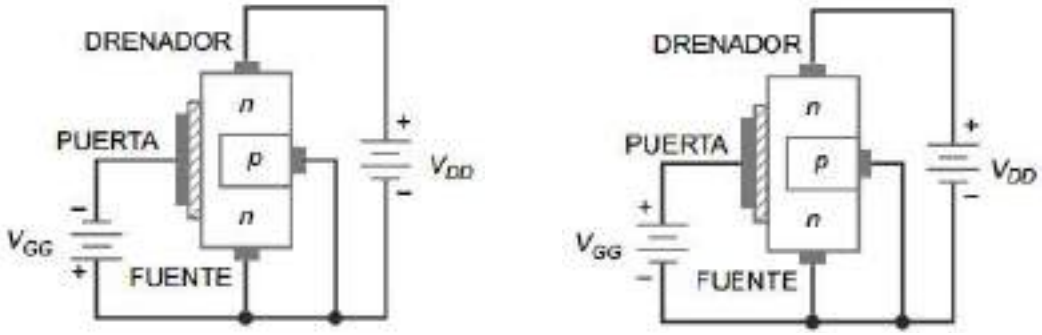


Figura 53: MOSFET en modo de vaciamiento.

A la puerta de un D-MOSFET se puede aplicar una tensión negativa o positiva. Cuanto más negativa es la tensión de puerta (figura izquierda), más pequeña es la corriente de drenador. Cuando la tensión de puerta es lo suficientemente negativa,

la corriente de drenador se interrumpe. La tensión de puerta positiva aumenta el número de electrones libres que atraviesan el canal. Cuanto más positiva es la tensión de puerta, mayor será la conducción desde la fuente hacia el drenador.



(a) D-MOSFET con puerta negativa (b) D-MOSFET con puerta positiva

Figura 54: Polarización del MOSFET

Un E-MOSFET se muestra en la siguiente figura a la izquierda. El sustrato p ahora se extiende hasta tocar el dióxido de silicio. Como podemos ver, ya no hay un canal n entre la fuente y el drenador. El funcionamiento se explica con el E-MOSFET polarizado (figura a la derecha): Cuando la tensión de puerta es cero, la corriente entre la fuente y el drenador es cero. La única forma de obtener una corriente es con una tensión de puerta positiva. Cuando la tensión de puerta se hace positiva, atrae electrones libres a la región *p*. Cuando la tensión de puerta es lo suficientemente positiva, todos los huecos que tocan el dióxido de silicio están llenos y los electrones libres comienzan a fluir desde la fuente hasta el drenador. El efecto es el mismo que el de crear una delgada zona de material de tipo n junto al dióxido de silicio, que se denomina capa de inversión de tipo *n*.

(a) MOSFET en modo de enriquecimiento (b) E-MOSFET polarizado

Figura 55: E-MOSFET

La tensión V_{GS} mínima que crea la capa de inversión de tipo n se denomina tensión de umbral y se designa como V_{GS} (umbral). La siguiente figura muestra

un conjunto de curvas de drenador de un E-MOSFET de pequeña señal típico. La parte casi vertical de la gráfica es la región óhmica, y las partes casi horizontales definen la región activa.

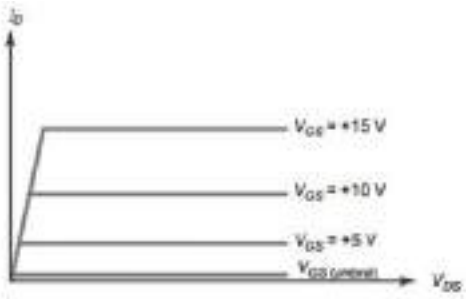
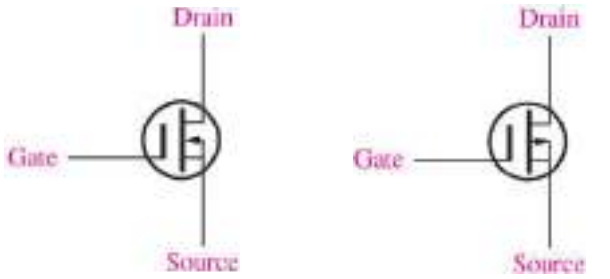


Figura 56: Curvas del drenador de un E-MOSFET.

Al igual que en casos anteriores, un E-MOSFET puede tener un canal (sustrato) de tipo p o de tipo n. El símbolo de esquema de ambos transistores se muestra en la figura a continuación (Malvino & Bates, 2007).



(a) E-MOSFET de canal *n* (b) E-MOSFET de canal *p*

Figura 57: Simbología del MOSFET.

Circuitos con transistores JFET y MOSFET

Los JFET tienen muchas aplicaciones. Los amplificadores JFET permiten controlar la corriente del colector con un voltaje de entrada muy pequeño. Por ejemplo, en la figura siguiente, se muestra un **amplificador en fuente común**. Toda la tensión alterna de entrada aparece entre la puerta y la fuente, lo que da lugar a una corriente alterna de drenador. Puesto que la corriente alterna de drenador fluye a través de la resistencia de drenador, obtenemos una tensión alterna de salida amplificada e invertida.

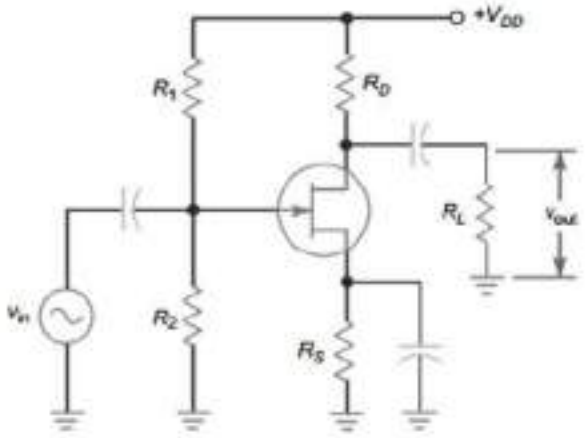


Figura 58: Circuito amplificador de fuente común.

Otra aplicación del JFET es la de **multiplexación**, es decir la combinación de varias señales en una sola línea. La figura siguiente muestra un multiplexador, un circuito que admite una o más señales de entrada y las pasa a la línea de salida. Cada JFET se comporta como un conmutador serie. Las señales de control (V1, V2 y V3) hacen que los JFET conduzcan y se corten (Malvino & Bates, 2007).

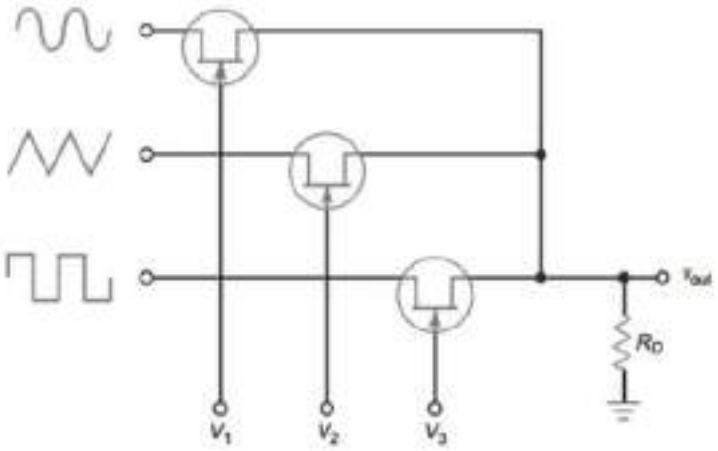
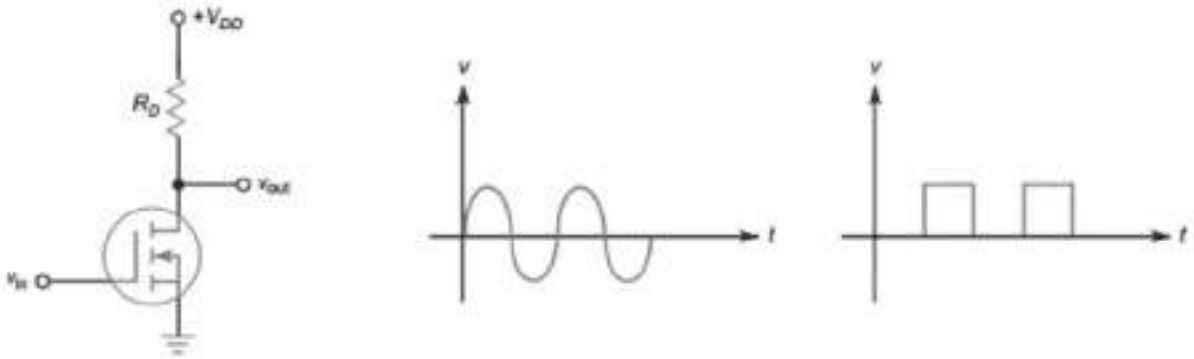


Figura 59: Circuito multiplexador

En el caso de los MOSFET, su uso (principalmente los E-MOSFET) ha permitido el vertiginoso desarrollo de las computadoras.

Uno de los usos más frecuentes es en la conmutación digital. Puesto que un MOSFET permite o bloquea el paso de corriente dependiendo de la polarización de la puerta, puede utilizarse para conmutar una señal analógica (continua) en una señal digital (cuadrada). La figura siguiente a la izquierda muestra un circuito de un E-MOSFET con carga pasiva. La tensión de entrada en la puerta V_{in} es una onda sinodal que puede ser un nivel bajo o alto. Cuando V_{in} es un nivel bajo, el MOSFET está en corte y Vout es igual a la tensión de alimentación V_{DD}. Cuando V_{in} es un nivel alto, el MOSFET se satura y Vout cae a un valor bajo.



(a) E-MOSFET con carga pasiva

(b) Señal analógica de entrada y señal de salida en Vout

Figura 60: Aplicaciones con e-MOSFET.

Un **CMOS** (Complementary MOS) utiliza dos MOSFET (o simplemente MOS) complementarios, uno de canal n y otro de canal p. En un CMOS, la señal que se quiere conmutar se conecta a las puertas de los transistores MOS, mientras que las fuentes y drenadores se conectan a una fuente o a masa. La figura a continuación muestra un ejemplo. El valor de Vout dependerá de la tensión (alta o cero) que entre a las puertas de cada MOSFET, en un caso permitirá el paso de la tensión, mientras el otro transistor queda abierto; y en el otro se conectará a masa.

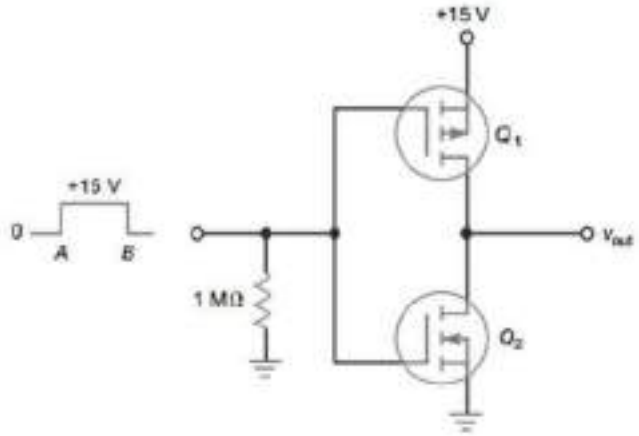


Figura 61: Símbolos de transistores MOS utilizados para formar un CMOS.

Un ejemplo del uso de CMOS es el inversor CMOS en un circuito digital básico (tema que veremos más adelante en otra unidad). Los dispositivos CMOS presentan la ventaja de tener un muy bajo consumo de potencia (Malvino & Bates, 2007).

VIDEO

En este video puedes revisar de nuevo cómo funcionan los transistores JFET y MOSFET, así como circuitos de aplicación de este tipo de transistores: https://youtu.be/Ofp__qry5fE?t=300.

Bibliografía

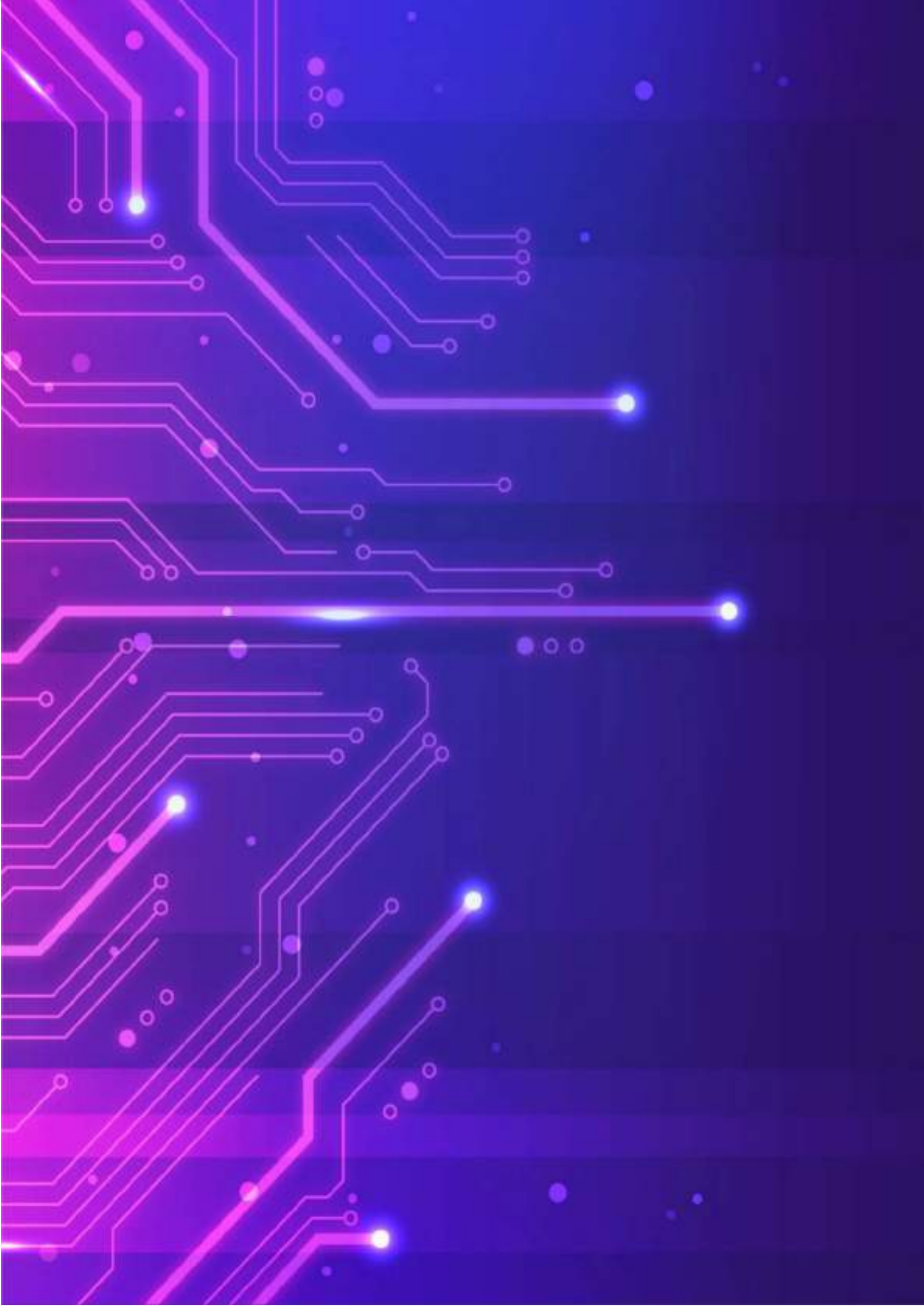
Dorf, R. C., & Svoboda, J. A. (2018). *Introduction to Electric Circuits.* Wiley.

Floyd, T. L. (2018). *Electronic Devices.* Pearson Education.

Graf, R. F. (1999). *Modern dictionary of electronics.* Butterworth-Heinemann: Newnes.

Malvino, A., & Bates, D. J. (2007). *Principios de Electrónica.* Madrid: McGraw-Hill.

Monk, S. (2017). *Electronic Cookbook, Practical Electronic Recipes with Arduino & Raspberry PI.* O'Reilly Media, Inc.



UNIDAD 02

Principios de Diseño Lógico Digital

Principios de Diseño Lógico Digital

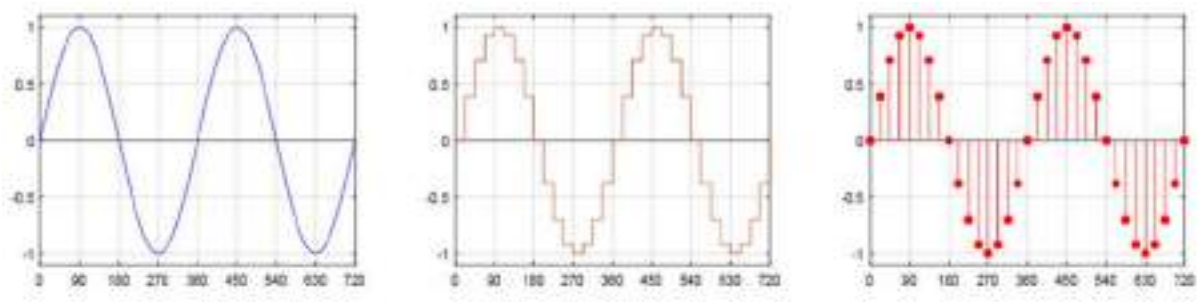
TEMA 2.1: INTRODUCCIÓN AL DISEÑO LÓGICO DIGITAL

En la unidad 1 aprendimos sobre los fundamentos y principios de la *electrónica analógica* de una manera muy rápida y básica, recorriendo desde sus orígenes, pasando por los conceptos de semiconductores, y llegando a los diodos y transistores, pero todo esto es sólo el comienzo de la aventura que viene a continuación. Ahora pasemos del 0 al 1.

2.1.1 Pasando del 0 Al 1

Como mencioné antes la electrónica vista hasta ahora es la **electrónica analógica** y la que estudiaremos en esta unidad será la **electrónica digital**, pero ¿cuál será la diferencia?

Muy bien, primero debemos entender dos conceptos **señales analógicas o continuas** y **señales digitales o discretas**, creo que ya vas entendiendo por donde va todo esto, necesito que te fijes muy bien en la onda seno generadas en la Figura 2-1, donde la Figura 2-1a es una señal de tipo analógica o tiempo continua, Figura 2-1b es una señal de tipo digital y Figura 2-1c es de tiempo discreta y es que la **electrónica analógica** solo puede trabajar con **señales analógicas**, y la **electrónica digital** solo puede trabajar con **señales digitales**, también llamadas **señales de tiempo discreto** por su forma de adquisición por muestreo.



(a) (b) (c)

Figura 2-1: Representación gráfica de la función seno como señal
(a) analógica o continua, (b) digital y (c) discreta.

Veamos un ejemplo que permitirá comprender mejor estas diferencias: si usted está conduciendo un vehículo entre dos ciudades, usted sabe que es

imposible que el vehículo viaje a una velocidad constante de 70 km/h, porque hay situaciones donde se debe subir y bajar la velocidad, incluso llegar a detener completamente el vehículo, pues bien, todos estos cambios de velocidad, usted puede observarlos durante el trayecto en el velocímetro, el cual los mide de forma continua, es decir, analógica, porque usted puede ver como la aguja del velocímetro fluctúa entre la velocidad inicial y la velocidad final pasando por varios valores durante esta fluctuación. Y como usted sabe esto, desea hacer un estudio de estos cambios de velocidad durante el viaje, y decide llevar un registro de las velocidades, tomando muestras instantáneas cada 2 minutos, a este proceso se le llama **muestreo**, porque usted está llevando un muestreo de la señal y este proceso debe realizarlo de forma discreta y discontinua, sin importarle los valores intermedios entre la velocidad 1 y la velocidad 2, por lo tanto, este registro al momento de ser graficado, se verá como una señal "**digital**". Ahora, conociendo este ejemplo, es muy probable que en usted puedan existir algunas dudas, por ejemplo:

- ¿Las señales digitales pueden perder información?
- ¿A caso las señales analógicas tienen más fidelidad que las señales digitales?
- ¿Cuál sería más recomendable?, etc.

Pues la verdad es que, si existen perdidas en las señales digitales y las analógicas tienen más fidelidad, pero hoy en día las brechas existentes han sido reducidas por los avances tecnológicos, porque contamos con más potencia de procesamiento y mejores métodos de almacenamiento, porque para solucionar estos "inconvenientes" se debe analizar el número de muestras que se toman del evento a analizar, porque mientras más muestras se tomen en un determinado tiempo, más exacta es la información obtenida y más fiel a la realidad es, esto puede observarlos en la calidad de las fotografías, la música, el vídeo, etc. Por otro lado, la implementación de equipos que procesan información digital es más simple y barato que implementar equipos que procesen información analógica. Además, trabajar con señales digitales tiene también sus ventajas, por ejemplo:

- Los datos digitales pueden ser procesados y transmitidos de forma más fiable y eficiente que los datos analógicos, porque la señal es más limpia.
- Se pueden almacenar de forma más eficiente.
- El ruido no afecta "tanto" a los datos digitales como a los datos analógicos.

2.1.2 Sistemas de numeración

Existen varios sistemas de numeración, los cuales no son otra cosa que formas de contar, de los cuales los más utilizados en sistemas digitales y la informática son los sistemas de numeración: binario, octal, decimal y hexadecimal. Y de estos cuatro sistemas de numeración utilizados, el más conocidos por nosotros es el sistema de numeración decimal.

- El **sistema de numeración decimal** se llama así por estar formado por los 10 dígitos (números) que todos conocemos: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9; a estos números se los conoce como números de base 10.
- El siguiente sistema de numeración es realmente la base de los sistemas digitales y la informática, estoy hablando del **sistema de numeración binario**, donde su prefijo "BI" que significa doble o dos, nos indica que este sistema de numeración cuenta únicamente con 2 dígitos: 0 y 1. Cuyo origen se basa en el concepto de encendido y apagado, donde el 0 significa apagado o sin energía y el 1 significa encendido. Pero, debido a complejidad de trabajar con números binarios muy extensos, realizar operaciones e incluso llegar a memorizarlos, se juntaron en grupos de 3 y 4 dígitos, a los cuales se los llamó **sistema de numeración octal** y **hexadecimal** respectivamente. Cuando veamos las tablas de verdad se entenderá este concepto. A estos números se los conoce como números de base 2 o números binarios.
- El **sistema de numeración octal** se llama así por estar formado por 8 números: 0, 1, 2, 3, 4, 5, 6 y 7; a estos números se los conoce como números de base 8.
- El **sistema de numeración hexadecimal** se llama así por estar formado por 16 números: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F; a estos números se los conoce como números de base 16 o H (por hexadecimal).

Una característica importante de los sistemas de numeración revisados previamente es el hecho de ser sistemas de numeración posicionales, es decir, según su base y la posición del número, este tiene su valor, ejemplo:

Número (1527_{10}):	1	5	2	7
Posición:	3	2	1	0
Pesos:	10^3	10^2	10^1	10^0
	$1*10^3$	$5*10^2$	$2*10^1$	$7*10^0$
	1000	500	20	7

Número 1527 de base 10 analizado según sus pesos.

Número (1011_2):	1	0	1	1
Posición:	3	2	1	0
Pesos:	10^3	10^2	10^1	10^0
	$1*10^3$	$0*10^2$	$1*10^1$	$1*10^0$
	1000	0	10	1

Número 1011 de base 2 analizado según sus pesos.

Número (1527_8):	1	5	2	7
Posición:	3	2	1	0
Pesos:	10^3	10^2	10^1	10^0
	$1*10^3$	$5*10^2$	$2*10^1$	$7*10^0$
	1000	500	20	7

Número 1527 de base 8 analizado según sus pesos.

Número ($A25F_H$):	A	2	5	F
Posición:	3	2	1	0
Pesos:	10^3	10^2	10^1	10^0
	$A*10^3$	$2*10^2$	$5*10^1$	$F*10^0$
	A000	200	50	F

Número A25F de base 16 analizado según sus pesos.

Una pregunta que puede resultar al ver los ejemplos anteriores es ¿por qué en cada caso la base de los pesos es 10 y no el valor de la base del sistema de numeración? Respuesta: según la base, el número que corresponde a la base es el 10, más claro, si utilizo la base 8, después del 7 no viene el 8, porque sus únicos dígitos son del 0 al 7, por lo tanto, en número que viene en lugar del 8 es el 10, lo

mismo para las bases 2 y 16, y también pasa en la base 10, solo que en este caso si coincide porque después del 9 como sabemos viene justamente el 10.

Es fundamental saber cómo realizar la conversión entre un sistema y otro. Sin embargo, ese es un tema que debió haber sido visto en una asignatura previa, por lo que no se profundizará mucho al respecto.

Como explicaba antes el trabajar con los números del **sistema de numeración binaria** es muy complicado, se agruparon en grupos de 3 y 4 dígitos para facilitar la utilización de estos números creando así los **sistemas de numeración octal** y **hexadecimal**, porque en el caso del **sistema de numeración octal** sus números del 0_8 al 7_8 en base 2 son: 000_2, 001_2, 010_2, 011_2, 100_2, 101_2, 110_2 y 111_2; y en el **sistema de numeración hexadecimal** sus números 0_H al F_H en base 2 son: 0000_2, 0001_2, 0010_2, 0011_2, 0100_2, 0101_2, 0110_2, 0111_2, 1000_2, 1001_2, 1010_2, 1011_2, 1100_2, 1101_2, 1110_2 y 1111_2 pero el **sistema de numeración decimal** no se ajusta a estos grupos, por lo cual es necesario utilizar el **código decimal binario** (**BCD**, *Binary Coded Decimal*) donde es una forma de expresar cada uno de los dígitos decimales con un código binario (hexadecimal) de 4 dígitos, tomando únicamente los códigos equivalentes para los números del 0 al 9 y dejando de utilizar los códigos equivalentes del 10 al 15, considerando estos últimos como inválidos.

Tabla 2-1: Conversión BCD

Decimal:	0	1	2	3	4	5	6	7	8	9
BCD:	0000_2	0001_2	0010_2	0011_2	0100_2	0101_2	0110_2	0111_2	1000_2	1001_2

2.1.3 Codificación binaria y niveles lógicos

En la sección más atrás cuando se habló de los números binarios, se explicó brevemente que su origen se debe a uno de dos posibles estados, encendido representado por el número "1" y apagado que está representado por el número "0", esta representación es comúnmente representada en los botones, ver Figura 2-2. A estos números se les denomina dígito binario o "*bit*" (por su nombre en inglés "**BI**nary digi**T**").

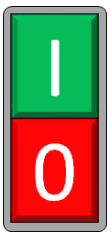


Figura 2-2: Representación de un interruptor de encendido y apagado.

Las computadoras modernas según su arquitectura reciben estos *bits* en grupos de: 4-btis llamado "*Nibbles*", 8-bits llamado "*Bytes*", 16-bits (2 bytes) llamado "Palabra" (WORD), 32-bits (4 bytes) llamado "Palabra Doble" (DWORD), y 64-bits (8 bytes) llamado "Palabra Cuádruple" (QWORD). Estas agrupaciones se utilizan para enviar instrucciones o señales al procesador, además de ser utilizadas para trabajar con números o caracteres usando los códigos binarios y ASCII respectivamente.

En la electrónica digital estos números binarios "0" y "1" representan un valor de voltaje o simplemente un nivel de voltaje, donde normalmente:

- “1” significa un nivel de voltaje alto (V_H), donde la “H” proviene de la definición en inglés “HIGH”.
- “0” significa un nivel de voltaje bajo (V_L), donde la “L” proviene de la definición en inglés “LOW”.

Puse “normalmente”, porque a esta definición o convención se le llama “**Lógica Positiva**”, pero también se da el caso de una “**Lógica Negativa**”, donde el “0” representa el nivel de voltaje alto y “1” representa el nivel de voltaje bajo. A estos niveles de voltaje alto y bajo se les denomina “**Niveles Lógicos**”. Estos niveles de voltaje suelen ser enviados a los circuitos digitales a través de una señal llamada “***tren de pulsos***”, la cual no es otra cosa que una sucesión de 1s y 0s. El ancho de estos pulsos va a depender de una señal de control periódica llamada reloj (ver Figura 2-3).

Estos trenes de pulsos son señales en serie, pero a través de un bus de datos pueden transmitirse en paralelo.

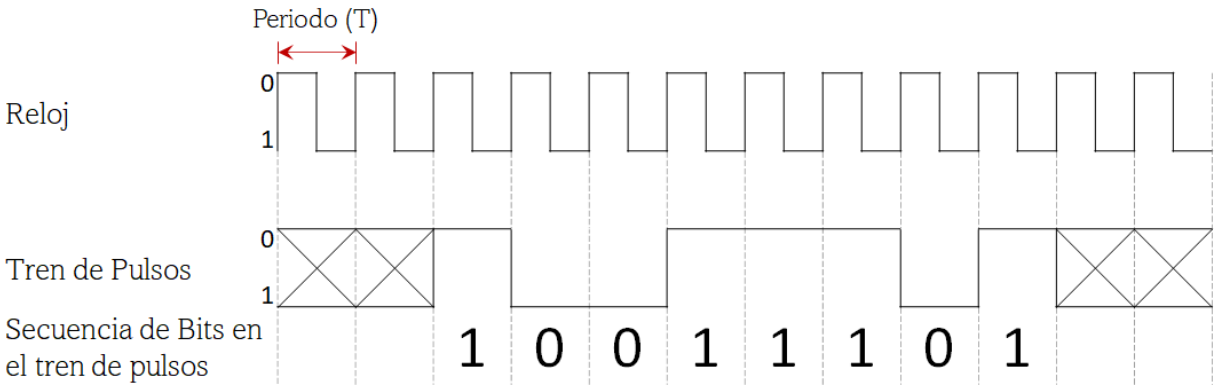


Figura 2-3: Ejemplo de una señal de reloj y un tren de pulsos, donde el periodo (T) del reloj indica la duración de los pulsos dentro del tren.

¿Cómo interpretar los pulsos?

Los pulsos como se ve en la Figura 2-3, pueden representar un “1” o un “0”, donde el pulso con el “1” se llama, en este caso, “**pulso positivo**” y el “0” se llama **“pulso negativo**”.

(a) (b)

Figura 2-4: Representación gráfica de los pulsos (a) positivo “1” y (b) negativo “0”

Para implementar los circuitos digitales, vamos a utilizar herramientas como compuertas lógicas, tablas de verdad, algebra booleana y mapas de Karnaugh (mapas K).

Como se habló previamente, las computadoras modernas agrupan estos números binarios en grupos de 2, 4, 8, 16, 32, 64 y 128 bits, para realizar operaciones de suma y restas, pero existe un problema con las restas o sumas con números negativos, porque en los sistemas normales podemos expresar el signo negativo (–) antes de un número para indicar su naturaleza, pero esto no es posible en los sistemas computacionales, porque las computadoras solo entienden “1” y “0”, no entienden de signos y mucho menos de números negativos, para lo cual se crearon dos métodos el primero expresando el bit más significativo como signo, donde “0” es positivo y “1” negativo, y el complemento a dos.

Primero les voy a explicar cómo se forma un numero binario y qué significan los conceptos de “Bit Más Significativo” (**MSB**, por su nombre en inglés “*More Significative Bit*”) y “Bit Menos Significativo” (**LSB**, por su nombre en inglés “*Less Significative Bit*”). Entonces, como expliqué antes, para enviar mensajes que puedan ser entendidos por el computador, es necesario enviarlos agrupados en un arreglo de n bits ordenados de izquierda a derecha (por su forma de leerlos), donde, el primer bit, llamado también “Bit 0” o “LSB” es el bit ubicado más a la derecha del arreglo, y el último bit, llamado “Bit n-1” o MSB es el bit ubicado más a la izquierda del arreglo. En la siguiente figura se muestra como está estructurado un dato o arreglo binario de 8 bits (también llamado byte), los números formados se leen al igual que los de base decimal de izquierda a derecha. Es importante detallar que tener los bits en grupos de 2, 4, 8, 16, 32, 64, etc.; si el numero escogido no llena el espacio reservado por ser muy pequeño, a la izquierda de este debe ser completado por ceros “0”, en los siguientes ejemplos se podrá apreciar esto.

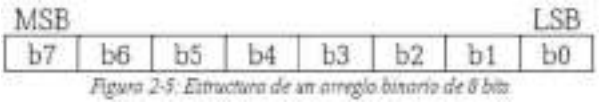


Figura 2-5: Estructura de un arreglo binario de 8 bits

Figura 2-5: Estructura de un arreglo binario de 8 bits

Para tenerlo un poco más claro, dependiendo de la arquitectura del computador y de su procesador, será el tamaño de la agrupación de bits, tanto así que, si un computador tiene un procesador de 8 bits utilizaría arreglos de 8 bits para procesar sus datos, de la misma forma que un computador que use un procesador de 64 bits, lo hará con arreglos de 64 bits, y así para los otros procesadores de 16, 32 y 128 bits. De ahí que enviar un dato al computador, por ejemplo, el número 539 de base 10 en binario (1000011011) dependerá de su procesador y su estructura sería:

- **16 bits:**
  00000010 00011011
- **32 bits:**
  00000000 00000000 00000010 00011011
- **64 bits:**
  00000000 00000000 00000000 00000000 00000000 00000000 00000010 00011011

Como se puede ver, el número 539 en binario solo tiene 12 bits, por lo tanto, es necesario completar con “0” es espacio restante, para evitar que datos erróneos o basura lleguen al procesador y genere procesos erróneos. Además, note usted que mientras más número de bits utilice el procesador, hace que sea más complicado trabajar los datos usando números binarios, es decir, lo hace muy complejo, de ahí que es más fácil utilizar el sistema hexadecimal para representar estos números, porque cada 4 bits (Nibble) se transformaría a este sistema. Ahora, revisando el ejemplo anterior, su equivalente en código hexadecimal sería:

- **16 bits:**
  $021B_H$
- **32 bits:**
  $0000\ 021B_H$
- **64 bits:**
  $0000\ 0000\ 0000\ 021B_H$

Números negativos

Hasta ahora solo hemos trabajado con números positivos, pero ¿Y los números negativos?, como expliqué antes, el computador solo interpreta “0” y “1”, y nosotros usamos el signo negativo para transformar un número positivo, pero el computador no lo hace, porque no lo conoce, entonces, si queremos expresar el número -539, no podemos solo escribirlo de la forma - 0000 0010 0001 1011 y listo, sino que se debe utilizar uno de los siguientes métodos:

Por Signo:

Este método utiliza el MSB para usarlo como un bit de signo, siendo “0” el positivo y “1” el negativo, y dejando los otros n-2 bits intactos. Aplicando el caso del ejemplo anterior tenemos:

- **16 bits:**
  10000010 00011011
- **32 bits:**
  10000000 00000000 00000010 00011011
- **64 bits:**
  10000000 00000000 00000000 00000000 00000000 00000000 00000010 00011011

Complemento a 2:

Este método se complementan todos los bits del arreglo (complementar: cambiar de 1 a 0 y de 0 a 1) y luego sumar 1 al resultado. Aplicando el caso del ejemplo anterior tenemos:

- **16 bits:**
  11111101 11100100 + 1 =
  11111101 11100101
- **32 bits:**
  11111111 11111111 11111101 11100100 **+ 1** =
  11111111 11111111 11111101 11100101
- **64 bits:**
  11111111 11111111 11111111 11111111 11111111 11111111 11111101 11100100 **+ 1** =
  11111111 11111111 11111111 11111111 11111111 11111111 11111101 11100101

Tema 2.2: Compuertas lógicas básicas, universales y equivalencias

Las compuertas lógicas son circuitos digitales muy sencillos que están formados por transistores, que según su construcción estos pueden ser de tipo CMOS (semiconductor complementario de óxido metálico) por usar transistores de tipo MOS que analizamos en la unidad anterior y los más usados de tipo TTL (lógica transistor a transistor) por usar transistores de tipo BJT, también analizados en la unidad anterior. En esta sección del capítulo, evitaremos entrar en detalles de su construcción y solo nos enfocaremos en el uso correcto de las compuertas. Cabe destacar que al ser circuitos electrónicos los voltajes altos y bajos no son valores tácitos, sino un pequeño rango del voltaje es que se considera alto o bajo, estos pequeños rangos de voltaje se deben al ruido que puede haber en los cables al cambiar de un nivel a otro, a estos rangos se les llama márgenes de ruido (ver Figura 2-6).

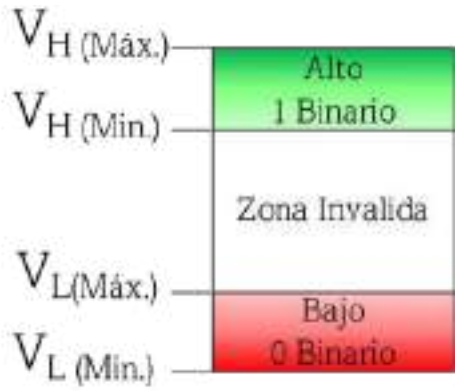


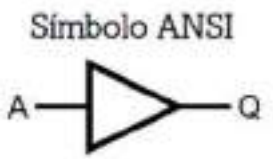
Figura 2-6: Márgenes de los niveles de voltaje para los niveles alto (verde) y bajo (rojo)

Las compuertas lógicas representan operaciones matemáticas como la suma y multiplicación a nivel binario. Las compuestas que revisaremos en este compendio son: Y, O, O exclusivo, búfer, y sus negaciones. Empecemos con la compuerta más básica de todas, el búfer, también llamado búfer digital.

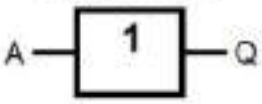
2.2.1 Búfer digital (Buffer digital)

El búfer digital es una compuerta que tiene una entrada y salida, este tipo de circuitos mantiene o copia el valor de la entrada en la salida, es decir, si tenemos un valor en alto (1) en la entrada (A) del búfer, a la salida (Q) de este, tendremos

el mismo valor en alto (1), como se explicó, su principal uso es mantenerla calidad de la señal digital o mejorarla.



Símbolo ANSI



Símbolo IEC

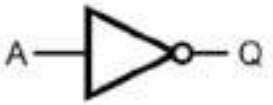
Ecuación

$Q = A$

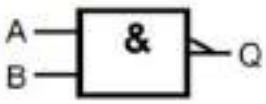
A	Q
0	0
1	1

2.2.2. Compuerta Negación Lógica (NOT Gate)

La compuerta de negación o simplemente llamada inversor al igual que el búfer digital tienen una sola entrada y salida, cuya función principal es negar el valor almacenado en la entrada de este, es decir: si a la entrada (A) se tiene un valor en alto (1), a la salida (Q) de la compuerta de inversión se tiene un bajo (0). Matemáticamente se suele representar la relación de la entrada y la salida del negador de las siguientes formas: $Q = \overline{A}\overline{A}$, Q = A′, Q = !A, Q = ~A, y Q = ¬A.



Símbolo ANSI



Símbolo IEC

Ecuación

$Q = \overline{A}$

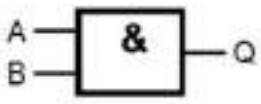
A	Q
0	1
1	0

Como se puede ver, la principal diferencia entre el símbolo del búfer y la negación es el circulo a la salida (○), este círculo que nos va a indicar en las compuertas que las salidas y/o entradas de estas están invertidas.

2.2.3 Compuerta “Y” Lógica (AND Gate)

La compuerta lógica “Y”, también llamada AND, se caracteriza por tener dos o más entradas y una sola salida, esta compuerta funciona de manera similar a la multiplicación, donde su principal condición radica en tener todas las entradas en alto, para tener la salida también en alto, caso contrario, si al menos una de las entradas está en bajo, la salida de esta compuerta estará automáticamente en bajo. Esta compuerta se puede definir como de tipo incluyente. Su función se podría leer como “todas o ninguna”. Matemáticamente se suele representar la relación de la entrada y la salida de la compuerta lógica “Y” de las siguientes formas: Q = AB, Q = A·B, Q = A ∧ B, y Q = A & B.

Símbolo ANSI



Símbolo IEC

Ecuación

$Q = AB$

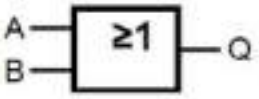
A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

2.2.4 Compuerta “O” Lógica (OR Gate)

La compuerta lógica “O”, también llamada OR, se caracteriza por tener dos o más entradas y una sola salida, esta compuerta funciona de manera similar a la suma, donde su principal condición radica en tener al menos una de las entradas en alto, para tener la salida también en alto, caso contrario, si todas sus las entradas están en bajo, la salida de esta compuerta estará automáticamente en bajo. Esta compuerta se puede definir como de tipo disyuntiva. Su función se podría leer como “al menos una”. Matemáticamente se suele representar la relación de la entrada y la salida de la compuerta lógica “O” de las siguientes formas: Q = A + B, Q = A ∨ B, y Q = A ‖ B.

Símbolo ANSI

Símbolo IEC



Ecuación

$$Q = A + B$$

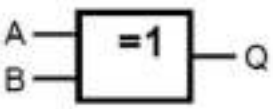
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	1

2.2.5 Compuerta “O-Exclusiva” Lógica (XOR Gate)

La compuerta lógica “O-Exclusiva”, también llamada XOR, se caracteriza por tener dos o más entradas y una sola salida, la principal función de esta compuerta radica en que únicamente la salida estará configurada en alto, si y solo si una de las entradas está en alto, caso contrario, si más de una de sus entradas están en alto o todas en bajo, la salida de esta compuerta estará automáticamente en bajo. Esta compuerta se puede definir como de tipo excluyente. Su función se podría leer como “solo una”. Matemáticamente se suele representar la relación de la entrada y la salida de la compuerta lógica “O-Exclusiva” de las siguientes formas: Q = A ⊕ B, y Q = A ⩒ B.

Símbolo ANSI

Símbolo IEC



Ecuación

$$Q = A \oplus B$$

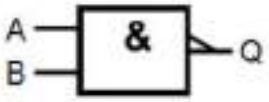
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

2.2.6 Compuerta “No Y” Lógica (NAND Gate)

La compuerta lógica “No Y”, también llamada NAND, se caracteriza por tener dos o más entradas y una sola salida, esta compuerta funciona de manera similar a la Y lógica, pero con la diferencia de tener su salida negada, de ahí que su principal condición radica en tener todas las entradas en alto, para tener la salida en bajo, caso contrario, si al menos una de las entradas está en bajo, la salida de esta compuerta estará automáticamente en alto. Esta compuerta equivale a tener una compuerta “Y” lógica seguida de una compuerta negación.

Símbolo ANSI

Símbolo IEC



Ecuación

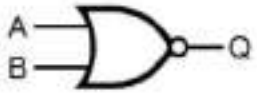
$$Q = \overline{A \cdot B}$$

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

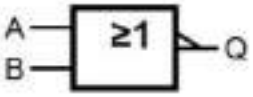
2.2.7 Compuerta "No O" Lógica (NOR Gate)

La compuerta lógica "No O", también llamada NOR, se caracteriza por tener dos o más entradas y una sola salida, esta compuerta funciona de manera similar a la O lógica, pero con la diferencia de tener su salida negada, de ahí que su principal condición radica en tener todas las entradas en bajo, para tener la salida en alto, caso contrario, si al menos una de las entradas está en alto, la salida de esta compuerta estará automáticamente en bajo. Esta compuerta equivale a tener una compuerta "O" lógica seguida de una compuerta negación.

Símbolo ANSI



Símbolo IEC



Ecuación

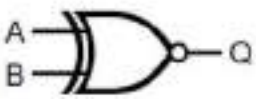
$$Q = \overline{A+B}$$

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

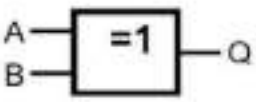
2.2.8 Compuerta No O-Exclusiva Lógica (XNOR Gate)

La compuerta lógica No O-Exclusiva se caracteriza por tener dos o más entradas y una sola salida, la principal función de esta compuerta radica en que únicamente la salida estará configurada en bajo, si y solo si una de las entradas está en alto, caso contrario, si más de una de sus entradas están en alto o todas en bajo, la salida de esta compuerta estará automáticamente en alto. Al igual que los casos anteriores, esta compuerta equivale a tener una compuerta "O-Exclusiva" lógica seguida de una compuerta negación.

Símbolo ANSI



Símbolo IEC



Ecuación

$$Q = \overline{A \oplus B}$$

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

Tema 2.3: Tablas de verdad

Hasta ahora cuando conocimos las compuertas, vimos el uso de unas tablitas de apoyo para conocer cómo funcionaban estas, incluso de un lado estaban las entradas a las compuertas y del otro lado la respuesta esperada por su comportamiento, estas tablitas son: las **Tablas de Verdad (TV)**. Las tablas de verdad no son otra cosa, que arreglos, que como su nombre lo dice, muestra valores de verdad de la composición de sus datos, esta consta básicamente de dos partes: la parte de la derecha que serían las condiciones de entrada y la de la izquierda que nos las posibles salidas en función de sus entradas. A continuación, se muestras un ejemplo de una tabla de verdad del operador lógico “Y” con su representación básica en forma de función a su izquierda y con la representación del operador a la derecha.

A	B	Y = f(A, B)
F	F	F
F	V	F
V	F	F
V	V	V

A	B	Y = A ∧ B
F	F	F
F	V	F
V	F	F
V	V	V

Como se puede apreciar en la tabla se utilizaron valores de “F” y “V” en lugar de “0” y “1”, esto se debe a sus significados de falso y verdadero respectivamente. Ahora existe un equivalente entre estas, porque “F” representa el “0” y “V” representa al “1”. En otras palabras, estas tablas de verdad son usadas explicar el funcionamiento de un circuito lógico, y el comportamiento que este tiene bajo ciertas configuraciones de entradas.

El principal uso de las tablas de verdad es poder diseñar circuitos de sistema digitales, porque conociendo las entradas y las salidas, puedo determinar qué condiciones debe haber en las entradas para una determinada salida.

Tema 2.4: Algebra booleana y el mapa de Karnaugh, simplificación y diseño de circuitos lógicos

En este tema aprenderemos a diseñar un circuito lógico después de haberlo analizado y simplificado con técnicas como el álgebra booleana y el mapa de Karnaugh (Mapa-K), pero primero.

2.4.1 Álgebra y ecuaciones booleanas

En el tema anterior de las tablas de verdad, ahora gracias a George Boole (izquierda) quien fue el inventor de esta álgebra, vamos a entender cómo usar realmente estas tablas de verdad y generar las **ecuaciones booleanas**, todo esto ayudado por un conjunto de reglas y axiomas que él desarrolló, las cuales caracterizan el comportamiento de la lógica matemática, de hecho esta álgebra se caracteriza por tener en sus resultados, sólo uno de dos posibles valores, "verdadero" o "falso", que para nuestro estudio de **Fundamentos de Sistemas Digitales** los reemplazaremos por sus equivalentes lógicos "1" y "0", Y por supuesto, debemos aceptar que todos estos axiomas característicos del Álgebra de Boole son verdaderos, ahora tenemos una variable B con dos posibilidades "0" y "1",

Figura 2-7: George Boole.

Tabla 2-2: Axiomas del Álgebra Booleana

Nombre	Axioma	Principio de Dualidad
Campo Binario	A1 $B = 0$ si $B \neq 1$	A1' $B = 1$ si $B \neq 0$
Negación (NOT)	A2 $\bar{0} = 1$	A2' $\bar{1} = 0$
Y/O	A3 $0 \cdot 0 = 0$	A3' $1 + 1 = 1$
Y/O	A4 $1 \cdot 1 = 1$	A4' $0 + 0 = 0$
Y/O	A5 $0 \cdot 1 = 1 \cdot 0 = 0$	A5' $1 + 0 = 0 + 1 = 1$

En base a los axiomas mostrados en la Tabla 2-2, se pueden usar 5 teoremas de una variable (Tabla 2-3) y 7 teoremas más, para cuando existen 2 o más variables (Tabla 2-4), los cuales junto a sus dualidades nos ayudarán a reducir o simplificar las ecuaciones booleanas. Por cierto, si aún no lo has notado, el principio de dualidad consiste en cambiar la "Y" por "O", es decir "•" por "+" y viceversa. Todos estos teoremas son verificables utilizando los axiomas.

Tabla 2-3: Teoremas Booleanos de una variable

Nombre	Teorema	Principio de Dualidad
Identidad	T1 $B \cdot 1 = B$	T1' $B + 0 = B$
Elemento Nulo	T2 $B \cdot 0 = 0$	T2' $B + 1 = 1$
Idempotencia	T3 $B \cdot B = B$	T3' $B + B = B$
Involución	T4 $\bar{\bar{B}} = B$	
Complementos	T5 $B \cdot \bar{B} = 0$	T5' $B + \bar{B} = 1$

Tabla 2 4: Teoremas Booleanos de varias variables

Nombre	Teorema	Principio de Dualidad
Conmutativa	T6 $B \cdot C = C \cdot B$	T6' $B + C = C + B$
Asociativa	T7 $(B \cdot C) \cdot D = B \cdot (C \cdot D)$	T7' $(B + C) + D = B + (C + D)$
Distributiva	T8 $(B \cdot C) + (B \cdot D) = B \cdot (C + D)$	T8' $(B + C) \cdot (B + D) = B + (C \cdot D)$
Absorción	T9 $B \cdot (B + C) = B$	T9' $B + (B \cdot C) = B$
Combinación	T10 $(B \cdot C) + (B \cdot \overline{C}) = B$	T10' $(B + C) \cdot (B + \overline{C}) = B$
Consenso	T11 $(B \cdot C) + (\overline{B} \cdot D) + (C \cdot D) = (B \cdot C) + (\overline{B} \cdot D)$	T11' $(B + C) \cdot (\overline{B} \cdot D) \cdot (C + D) = (B + C) \cdot (\overline{B} \cdot D)$
Teorema de De Morgan	T12 $\overline{B_0 \cdot B_1 \cdot B_2 \cdots B_n} = \overline{B_0} + \overline{B_1} + \overline{B_2} + \cdots + \overline{B_n}$	T12' $\overline{B_0 + B_1 + B_2 \cdots B_n} = \overline{B_0} \cdot \overline{B_1} \cdot \overline{B_2} \cdots \cdot \overline{B_n}$

Una vez conocido y definido el concepto de tablas de verdad y el álgebra booleana, pasamos a conocer dos tipos de operaciones que nos ayudarán a obtener la **ecuación booleana** característica de un sistema, la primera de estas operaciones es la “Suma de Productos”, llamada **SOP** (por su nombre en inglés *Sum of products*), la cual forma la ecuación al realizar la sumatoria de los productos (llamados mintérminos) cuyo valor sean verdadero en la TV, cuya representación matemática es: F(entradas) = ∑(mintérminos) y la segunda de estas operaciones es el “Producto de Sumas”, llamada **POS** (por su nombre en inglés *Product of Sums*), el cual forma la ecuación multiplicando las sumas (llamados maxtérminos) cuyo valor sean falso en la TV, cuya representación matemática es: F(entradas) = ∏(maxtérminos). Para entender cómo se usan los SOP y POS, primero debemos analizar lo siguiente.

Si tenemos una tabla de verdad con **n** entradas, tendríamos $2^n 2^n$ filas, una por cada posible combinación de las entradas. Para el ejemplo de la compuerta “Y” que estamos utilizando, se tendrían 2 entradas “A” y “B” y una salida “Q”, esto quiere decir que tenemos $2^2 2^2$ combinaciones de los valores de entrada, o sea 4, ahora para el caso de SOP cada uno de los 4 valores de la columna salida, está relacionada a la combinación de los elementos de las columnas de entrada y a esta relación se le llama mintérmino, Es decir que, si las entradas “A” y “B” tienen valores de 1 y 0 respectivamente, el mintérmino equivalente para estas entradas es Q = $A\overline{B}\overline{B}$, siempre y cuando el resultado de esta combinación de “A” y “B” sea verdadero, caso contrario sería “0”, donde la barrita superior en la “B”, significa complemento o negada, indicando que el valor en “B” es “0” y por eso, es necesario que el “0” se tratado como un “1”, para que la relación $A \cdot \overline{B}\overline{B}$, sea verdadera. En el caso de la ecuación formada mediante POS, mantenemos los valores de 1 y 0 para “A” y “B” del ejemplo anterior, el maxtérmino equivalente para esta relación sería Q = ($\overline{A}\overline{A}$ + B), siempre y cuando el resultado de esta combinación de “A” y “B”

sea falso, sino caso contrario sería “1”, ahora veámoslo más claro con el ejemplo de la compuerta lógica “Y”, pero ya no usando “F” y “V”, sino “1” u “0” en su lugar.

Ecuación SOP

A	B	Y= f(A, B)	mintérminos	Nombre del mintérmino
0	0	0	$\overline{A}\overline{B}$	m_0
0	1	0	$\overline{A}B$	m_1
1	0	0	$A\overline{B}$	m_2
1	1	1	AB	m_3

Resultado:

$$Y = \sum(m_3)$$
$$Y = AB$$

Ecuación POS

A	B	Y = f(A, B)	maxtérminos	Nombre del maxtérmino
0	0	0	$A+B$	M_0
0	1	0	$A+\overline{B}$	M_1
1	0	0	$\overline{A}+B$	M_2
1	1	1	$\overline{A}+\overline{B}$	M_3

Resultado:

$$Y = \prod(M_0, M_1, M_2)$$
$$Y = (A+B)(A+\overline{B})(\overline{A}+B)$$

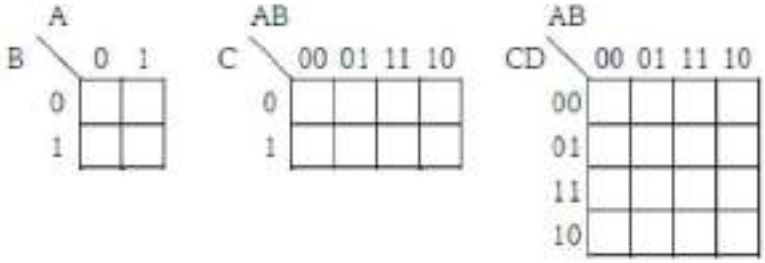
2.4.2 Mapa de Karnaugh

También llamados **Mapas-K**, es una herramienta para simplificar las ecuaciones booleanas de una manera eficiente, incluso, hay ocasiones donde una misma ecuación al ser simplificada usando las reglas del álgebra booleana llega a un punto donde matemáticamente ya no es más simplificable, pero con el mapa-k, se simplifica aún más.

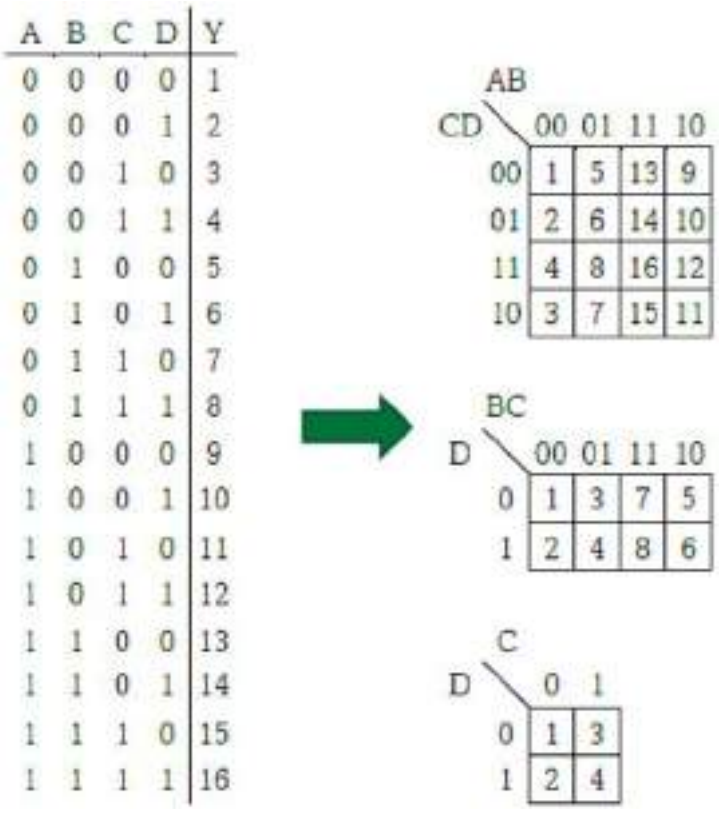
El mapa-k es una matriz cuyo tamaño depende del número de entradas del circuito, entonces si n es el número de las entradas de nuestro circuito, este genera $2^n 2^n$ combinaciones de sus entradas, y número de combinaciones definen el tamaño del mapa-k, de ahí que:

- 2 variables definen una matriz de 2x2 para 4 elementos.
- 3 variables definen una matriz de 3x2 para 8 elementos.
- 4 variables definen una matriz de 4x4 para 16 elementos.

Para 5 y 6 variables se recomienda utilizar 2 mapas-k de 4x4 por cada variable adicional, es decir 2 mapas-k para 5 variables y 4 mapas-k para 6 variables, para más de 6 variables se utilizan otros métodos. Los mapas-k ordenan las variables de entrada usando el código Gray: 00, 01, 11, 10. Veamos cómo se estructuran los mapas-k, de 2, 3 y 4 variables.



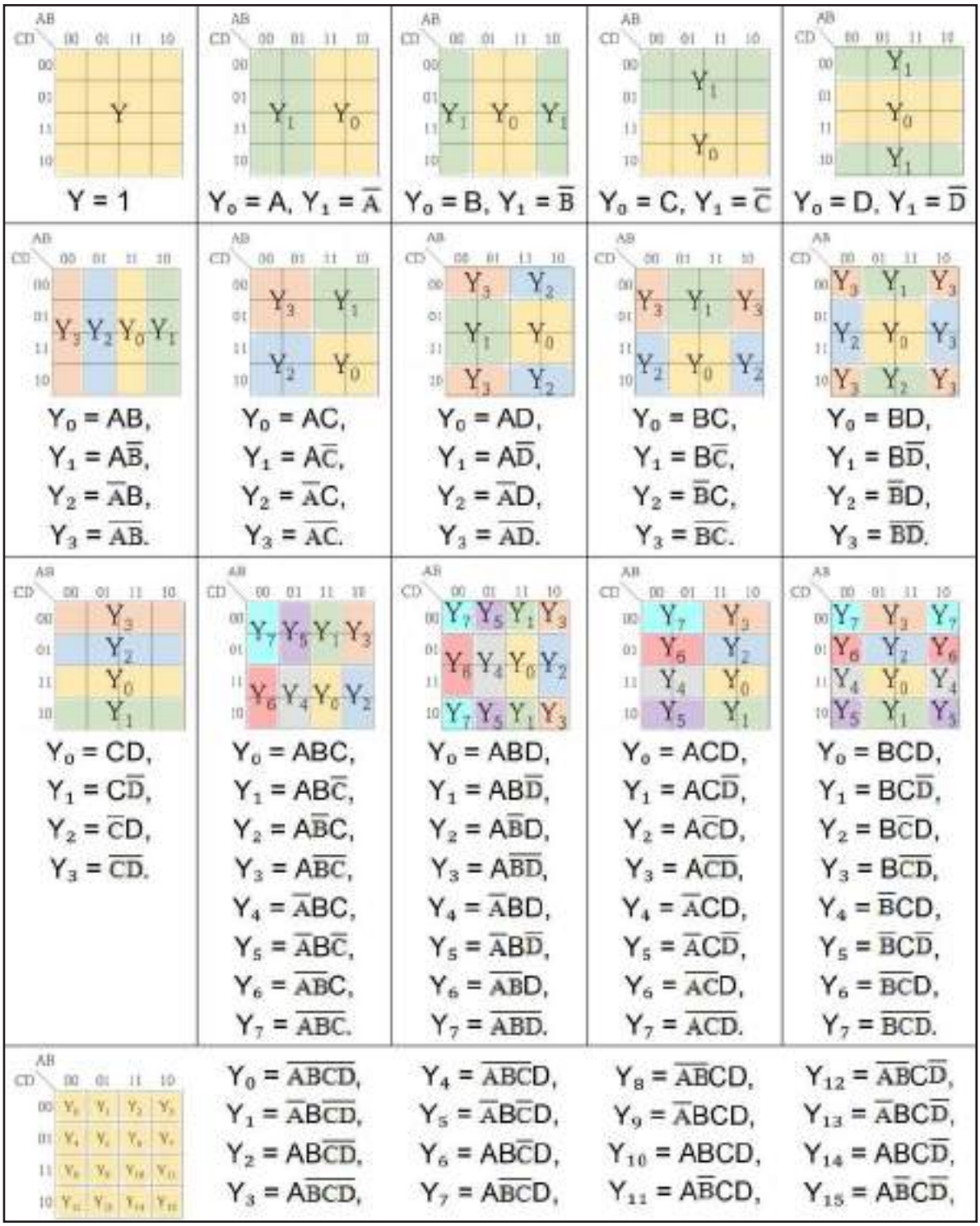
Una vez estructurado el mapa-k según el número de entradas, se procede a definir cuántos mapas-k se van a requerir, porque este número depende del número de salidas que tiene el circuito, entonces por cada salida del circuito, es necesario un mapa-k, de modo que al existir 2 salidas se necesitarán 2 mapas-k. Con todo ya definido veamos cómo se estructuran y se pasan los datos de la tabla de verdad al mapa-K. En la Figura 2-8, se muestra un ejemplo de cómo pasar y ordenar los datos de la tabla de verdad (en este caso de 2, 3 y 4 entradas y 1 salida) al mapa-k de 2x2, 3x2 y 4x4 respectivamente.



A	B	C	D	Y
0	0	0	0	1
0	0	0	1	2
0	0	1	0	3
0	0	1	1	4
0	1	0	0	5
0	1	0	1	6
0	1	1	0	7
0	1	1	1	8
1	0	0	0	9
1	0	0	1	10
1	0	1	0	11
1	0	1	1	12
1	1	0	0	13
1	1	0	1	14
1	1	1	0	15
1	1	1	1	16

Figura 2-8: Pasando datos de la tabla de verdad al mapa-k (2, 3 y 4 entradas)

Cuando los datos ya han sido pasados de la tabla de verdad al mapa-k, se procede a agrupar los “1” que se encuentren en el mapa-k, estos grupos son de 1, 2, 4, 8 y 16 elementos. A continuación, se presentan las posibles agrupaciones en un mapa-k de 4x4 y sus respectivo mintérminos.



En ocasiones puede aparecer casos de no me importa o indiferente a la salida de una ecuación

2.4.3 Diseño y simplificación de circuitos lógicos

Como se ha explicado hasta este punto, las TV son usadas principalmente para diseñar y/o evaluar un sistema digital, porque me permite ver cómo se comporta a la salida de este, de acuerdo a sus entradas, por lo tanto, este comportamiento en la salida está en función de las entradas y puede expresarse como una función o ecuación booleana.

Ejercicios:

1. Justificando los teoremas, simplificar la siguiente ecuación: $\bar{A}\bar{B}\bar{C}+A\bar{B}\bar{C}+A\bar{B}C$ $\bar{A}\bar{B}\bar{C}+A\bar{B}\bar{C}+A\bar{B}C$

Paso	Ecuación	Justificación
	$\bar{A}\bar{B}\bar{C}+A\bar{B}\bar{C}+A\bar{B}C$	
1	$\bar{B}\bar{C}(\bar{A}+A)+A\bar{B}C$	T8: Distributiva
2	$\bar{B}\bar{C}(1)+A\bar{B}C$	T5: Complemento
3	$\bar{B}(\bar{C}+AC)$	T1: Identidad y T8: Distributiva

2. Usando una tabla de verdad, pruebe el teorema de consenso y demuestre que la siguiente función $BC + \bar{B}D + CD = BC + \bar{B}DBC + \bar{B}D + CD = BC + \bar{B}D$ es correcta.

B	C	D	$BC + \bar{B}D + CD$	$BC + \bar{B}D$
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	1

La tabla de verdad indica que ambas funciones tienen el mismo resultado, por lo tanto, se comprueba el teorema de consenso.

Tema 2.5: Minimización de funciones lógicas.

El proceso que genera una expresión que contiene el menor número posible de términos con el mínimo número de variables posibles se denomina minimización. Para minimizar funciones lógicas se puede aplicar alguna de las estrategias que hemos visto en esta unidad.

Ejercicio:

1. Escribir una función o ecuación booleana en forma de suma de productos para la siguiente tabla de verdad y luego minimícela usando álgebra booleana y mapas-k. Implemente el circuito digital

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Suma de productos:

$Y = \overline{ABAB}C + AB\overline{CC} + ABC$

Minimización por álgebra de Boole

$Y = \overline{ABAB}C + AB(\overline{CC} + C)$

$Y = \overline{ABAB}C + AB(1)$

$Y = \overline{ABAB}C + AB$

Minimización por Karnaugh

C \ AB	00	01	11	10
0	0	0	1	0
1	1	0	1	0

$Y = \overline{ABAB}C + AB$

Implementación gráfica

Como se nota en la ecuación, hay la suma de dos mintérminos, esto quiere decir que debe utilizar una compuerta lógica de tipo “O”, luego analizando los mintérminos tenemos una compuerta lógica “Y” de tres entradas y otra de dos entradas. También vemos que dos de las entradas de la compuerta lógica “Y” están invertida, lo que significa que necesitaremos dos inversores. Entonces el circuito digital, quedaría así:

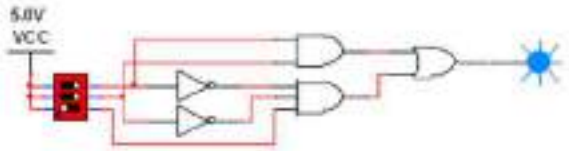


Figura 2-9: Esquemático del circuito digital.

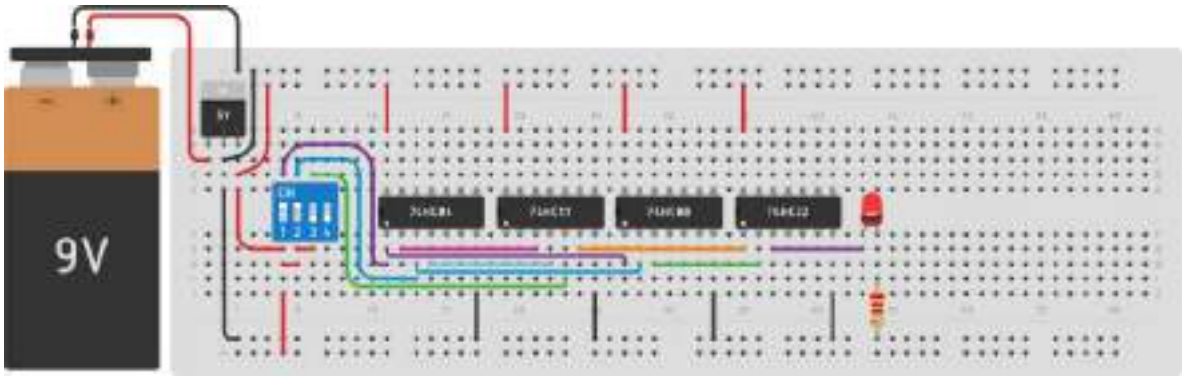


Figura 2-10: Implementación en TinkerCAD.

Tema 2.6: Análisis y síntesis de funciones lógicas.

Cuando ya se tiene claro lo que se desea implementar, se empieza a realizar un análisis lógico del circuito, donde a través de las tablas de verdad se determina que comportamiento deben tener las entradas del circuito, y como este debe reaccionar a su funcionamiento. Pongámoslo claro a través de un ejemplo:

Se tienen dos entradas, las cuales son dos sensores lógicos (S_1 y S_2), que se utilizan para controlar el nivel de un tanque de agua, estos sensores solo pueden enviar una señal al cuadro de mando cuando tienen la siguiente configuración: (0, 0), (0, 1) o (1, 1) y encender una señal luminosa (L), caso contrario, esta señal debería estar apagada.

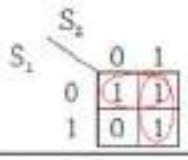
1. El primer paso por realizar es poner todo en una tabla de verdad

S1	S2	L
0	0	1
0	1	1
1	0	0
1	1	1

2. Se puede escribir una función SOP para L, dependiendo el comportamiento de S1 y S2

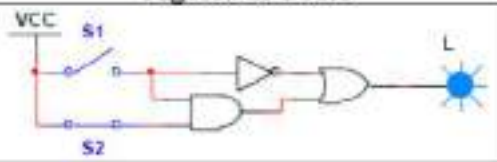
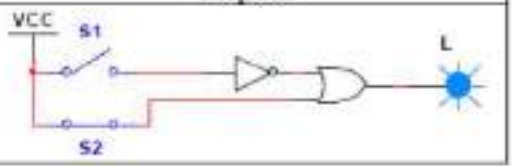
$$L = \overline{S_1}\,\overline{S_2} + \overline{S_1}S_2 + S_1S_2$$

3. Ver si es posible reducir la ecuación o función booleana usando cualquier de las herramientas álgebra de Boole o mapa-k

Álgebra de Boole	Mapa-k
$L = \overline{S_1}\,\overline{S_2} + \overline{S_1}S_2 + S_1S_2$ $L = \overline{S_1}(\overline{S_2} + S_2) + S_1S_2$ $L = \overline{S_1}(1) + S_1S_2$ $L = \overline{S_1} + S_1S_2$	
$L = \overline{S_1} + S_1S_2$	$L = \overline{S_1} + S_2$

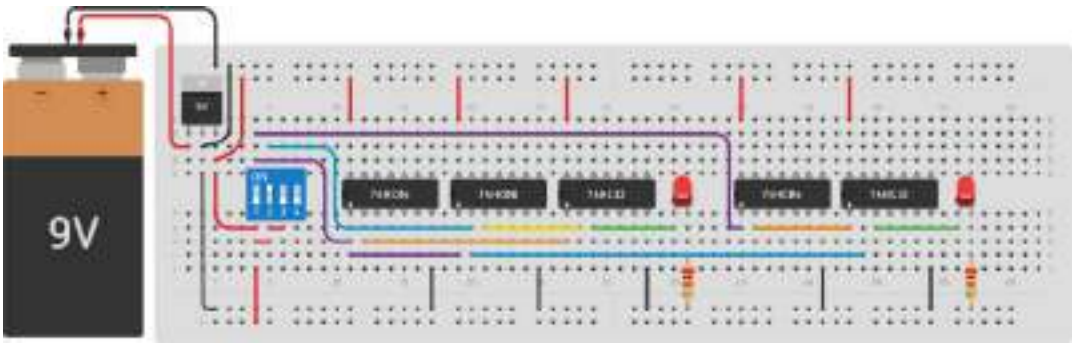
Como se pudo apreciar, la solución con el mapa-K es más reducida que la usada por el algebra de Boole

4. Diseñar

Álgebra de Boole	Mapa-k
	

Ambos diseños funcionan muy bien, pero en el circuito de la izquierda hay tres componentes lógicos y en el de la derecha solo 2, por lo tanto, el mapa-k optimizó mejor el circuito

5. Implementar y simular



GUÍA PRÁCTICA

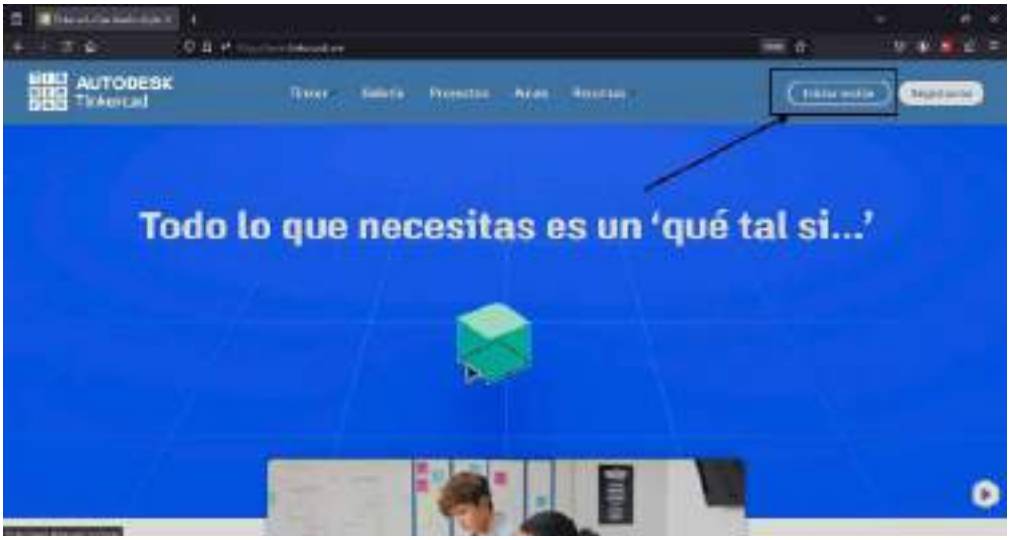
RECONOCIMIENTO DE LA INTERFAZ DE TINKERCAD PARA DISEÑAR CIRCUITOS

1. Introducción a Tinkercad

Tinkercad es una plataforma en línea gratuita desarrollada por Autodesk que permite diseñar y simular circuitos electrónicos de manera intuitiva. Es ideal tanto para principiantes como para usuarios avanzados que desean prototipar rápidamente.

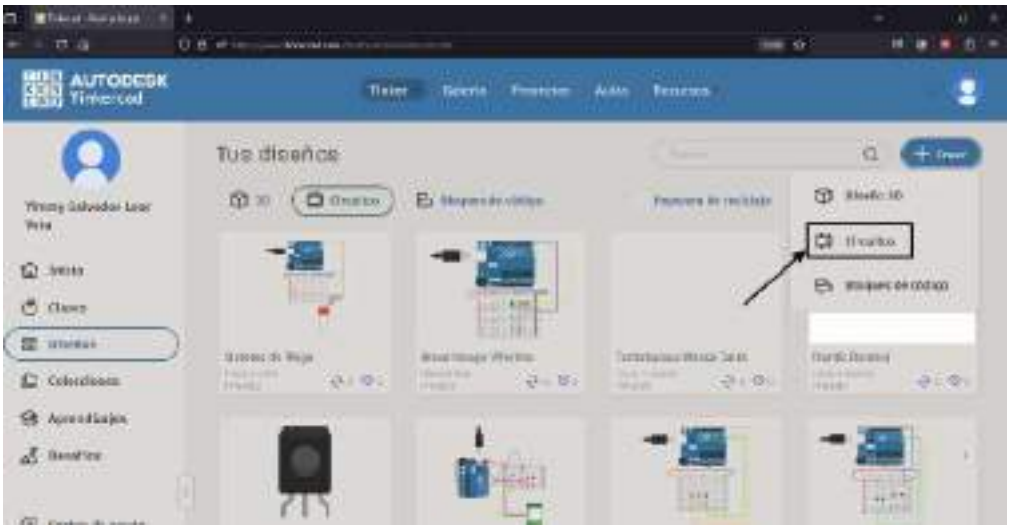
2. Creación de una Cuenta

- **Paso 1:** Visita www.tinkercad.com.
- **Paso 2:** Haz clic en **"Únete ahora"** o **"Join Now"**.
- **Paso 3:** Regístrate usando tu correo electrónico o una cuenta de Google/Apple.
- **Paso 4:** Confirma tu cuenta a través del correo electrónico si es necesario.



3. Accediendo al Módulo de Circuitos

- **Paso 1:** Inicia sesión en tu cuenta de Tinkercad.
- **Paso 2:** En el panel principal, encontrarás diferentes opciones: **"Diseño 3D"**, **"Circuitos"**, **"Codeblocks"**, etc.
- **Paso 3:** Haz clic en **"Circuitos"** para acceder al entorno de diseño de circuitos.



4. Explorando la Interfaz de Circuitos

A. Barra de Herramientas Superior

- **Nombre del Proyecto:** Puedes cambiar el nombre predeterminado haciendo clic en él (por ejemplo, “Circuito sin título” a “Mi Primer Circuito”).
- **Botones de Control:**
    - **Iniciar Simulación / Detener Simulación:** Para simular el funcionamiento del circuito.
    - **Exportar:** Permite exportar tu circuito en diferentes formatos.
    - **Compartir:** Opción para compartir tu proyecto con otros usuarios.

B. Panel de Componentes (Lado Derecho)

- **Categorías de Componentes:** Componentes básicos, entrada, salida, potencia, etc.
- **Barra de Búsqueda:** Para encontrar componentes específicos rápidamente.
- **Lista de Componentes:** Arrastra y suelta componentes como resistencias, LEDs, Arduino, etc., al área de trabajo.

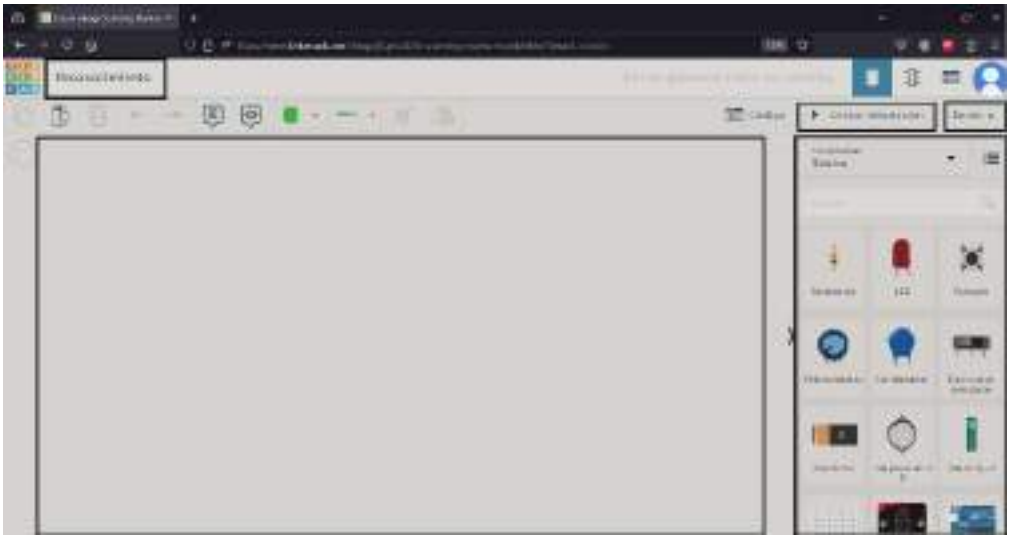
C. Área de Trabajo Principal

- **Lienzo Central:** Aquí es donde ensamblas tu circuito.
- **Herramientas de Zoom y Vista:** Acercar/alejar y ajustar la vista del circuito.

- **Configuración de la Vista:** Cambia entre vista en 2D y 3D si es necesario.

D. Panel de Propiedades

- **Al seleccionar un componente:** Aparece un panel donde puedes ajustar propiedades como valor de resistencia, color del LED, nombre del componente, etc.



5. Diseñando tu Primer Circuito

A. Añadir Componentes

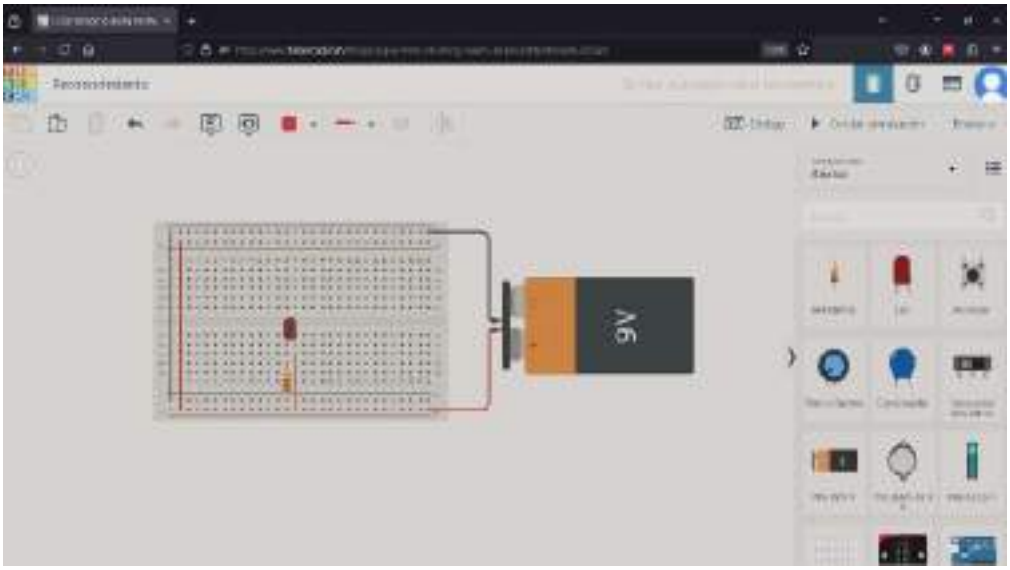
- **Paso 1:** Arrastra una **placa de pruebas (breadboard)** al área de trabajo.
- **Paso 2:** Añade una **fuente de alimentación** (por ejemplo, una batería de 9V).
- **Paso 3:** Coloca un **LED** y una **resistencia**.

B. Conectar Componentes

- **Paso 1:** Haz clic en un terminal del componente para iniciar una conexión.
- **Paso 2:** Arrastra el cursor hasta el punto de conexión deseado y haz clic para finalizar el cable.
- **Paso 3:** Puedes cambiar el color y la forma del cable para organizar mejor el circuito.

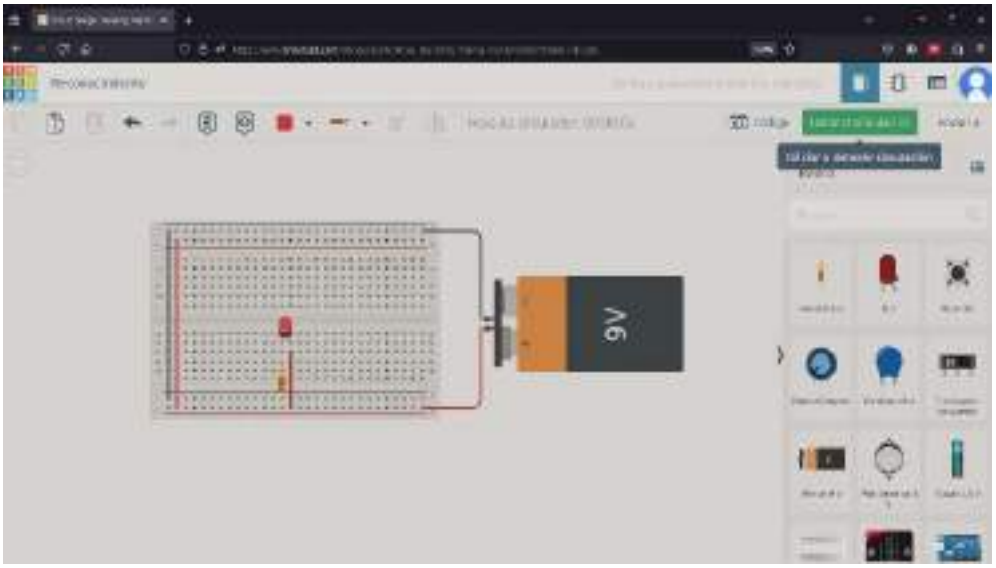
C. Ajustar Propiedades

- **Paso 1:** Selecciona la **resistencia** y ajusta su valor (por ejemplo, 220Ω).
- **Paso 2:** Selecciona el **LED** y elige el color que desees.



6. Simulación del Circuito

- **Paso 1:** Haz clic en **“Iniciar Simulación”** en la parte superior.
- **Paso 2:** Observa cómo el LED se enciende si el circuito está correctamente configurado.
- **Paso 3:** Si hay errores, aparecerán mensajes indicando problemas como cortocircuitos o conexiones faltantes



7. Guardar y Compartir tu Proyecto

- **Guardar:** Tinkercad guarda automáticamente los cambios en tu proyecto.
- **Renombrar Proyecto:** Haz clic en el nombre del proyecto para asignarle un nombre descriptivo.
- **Compartir:**
    - **Paso 1:** Haz clic en **"Compartir"**.
    - **Paso 2:** Elige si deseas generar un enlace público o invitar a colaboradores específicos.
    - **Paso 3:** Configura los permisos de acceso (solo ver o editar).

8. Recursos Adicionales

- **Tutoriales Integrados:** Accede a tutoriales paso a paso desde el panel principal de **"Circuitos"**.
- **Documentación y Ayuda:** Visita la sección de ayuda para obtener más información sobre funciones avanzadas.
- **Comunidad Tinkercad:** Únete a foros y grupos para compartir proyectos y obtener retroalimentación.

Consejos Finales

- **Explora:** Dedica tiempo a familiarizarte con los diferentes componentes y sus funciones.
- **Practica:** Intenta recrear circuitos básicos antes de pasar a diseños más complejos.
- **Aprende de Errores:** Utiliza los mensajes de error y advertencia para entender y corregir problemas en tus circuitos.
- **Mantén Organizado tu Trabajo:** Usa nombres claros y organiza tus proyectos en carpetas si es necesario.

PRÁCTICA

ENCENDER Y APAGAR UN LED CON ARDUINO

Objetivos:

- Familiarizarse con el uso de **Arduino** en Tinkercad.
- Controlar un **LED** a través de un código básico en **C++**.

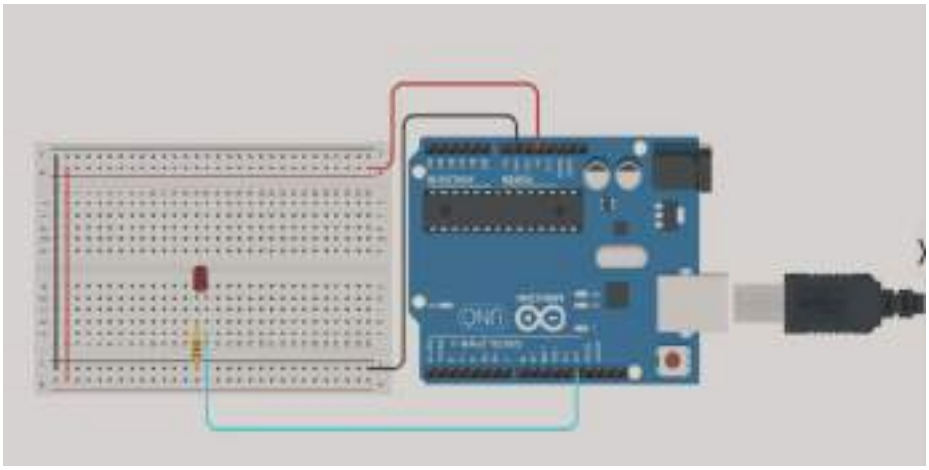
Materiales Necesarios:

- **1 x Arduino UNO**
- **1 x LED**
- **1 x Resistencia (220Ω)**
- **1 x Protoboard (Placa de pruebas)**
- **Cables de conexión**

Esquema de Conexión:

1. Conexión del LED:

- Conecta el **ánodo** (+, la pata larga) del LED a un **pin digital** de Arduino, por ejemplo, el **pin 13**.
- Conecta el **cátodo** (-, la pata corta) del LED a un extremo de la **resistencia de 220Ω**.
- El otro extremo de la **resistencia** se conecta a **GND** en la protoboard.



Paso a Paso en Tinkercad:

1. Abrir Tinkercad:

- Ve a Tinkercad e inicia sesión.
- En el panel principal, selecciona **“Circuitos”** y haz clic en **“Crear un circuito nuevo”**.

2. Añadir los Componentes:

- Arrastra un **Arduino UNO** desde el panel de componentes.
- Arrastra una **placa de pruebas (protoboard)**.
- Añade un **LED** y una **resistencia**.

3. Conexiones:

- Conecta el **ánodo** del LED al **pin 13** de Arduino.
- Conecta el **cátodo** del LED a una **resistencia de 220Ω**.
- Conecta la **resistencia** a **GND** en la placa de pruebas.

Código para Controlar el LED:

Una vez que hayas hecho las conexiones, sigue estos pasos para escribir el código.

1. Abrir el Editor de Código:

- En la parte superior derecha de la interfaz, selecciona **"Código"**.
- Cambia de **"Bloques"** a **"Texto"** para escribir el código en **C++**.

2. Escribir el Código:

```
// Declarar el pin del LED
int ledPin = 13;
void setup()
    {
    // Configurar el pin del LED como salida
    pinMode(ledPin, OUTPUT);
    }
    void loop()
    {
    // Encender el LED
    digitalWrite(ledPin, HIGH);
    delay(1000); // Esperar 1 segundo
    // Apagar el LED
    digitalWrite(ledPin, LOW); delay(1000); // Esperar 1 segundo
    }
```

Explicación del Código:

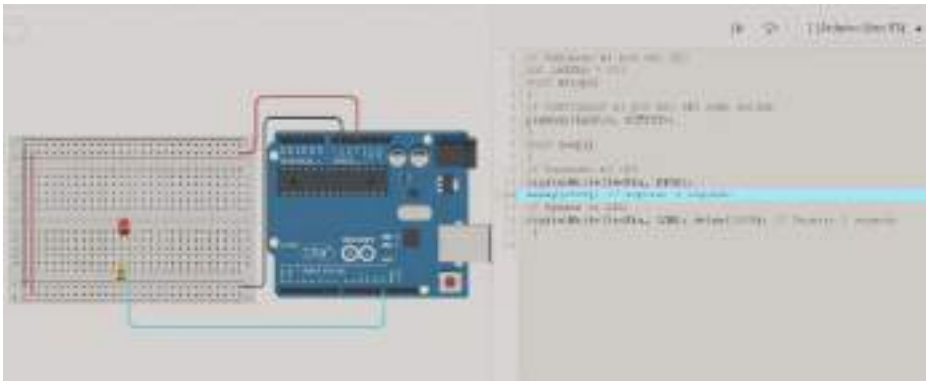
- pinMode(ledPin, OUTPUT);: Configura el pin 13 (donde está conectado el LED) como salida.
- digitalWrite(ledPin, HIGH);: Envía una señal de 5V al pin 13 para encender el LED.
- delay(1000);: Espera 1000 milisegundos (1 segundo) antes de continuar.
- digitalWrite(ledPin, LOW);: Envía una señal de 0V al pin 13 para apagar el LED.

- El ciclo se repite continuamente, encendiendo y apagando el LED con un intervalo de 1 segundo.

Simulación en Tinkercad:

1. Una vez que el circuito esté conectado y el código esté listo:

    - Haz clic en **"Iniciar Simulación"**.
    - El LED debería encenderse durante 1 segundo, apagarse durante 1 segundo, y repetir el ciclo.



Conclusión de la práctica:

Este ejercicio básico muestra cómo controlar un LED usando un Arduino a través de código. Es un excelente primer paso para familiarizarte con la programación de microcontroladores. Si deseas experimentar más, puedes ajustar los tiempos de encendido/apagado o controlar múltiples LEDs.

PRÁCTICA

SIMULACIÓN DE UN SEMÁFORO CON ARDUINO

Objetivos:

- Simular el funcionamiento de un semáforo utilizando tres LEDs (rojo, amarillo y verde).
- Programar los tiempos de encendido y apagado de cada luz del semáforo

Materiales Necesarios:

- **1 x Arduino UNO**
- **3 x LEDs (rojo, amarillo y verde)**
- **3 x Resistencias de 220Ω**

- **1 x Protoboard (Placa de pruebas)**
- **Cables de conexión**

Esquema de Conexión:

1. LED Rojo:

- Conecta el **ánodo** (pata larga) del LED rojo al **pin 13** de Arduino.
- Conecta el **cátodo** (pata corta) a una **resistencia de 220Ω**.
- El otro extremo de la resistencia se conecta a **GND**.

2. LED Amarillo:

- Conecta el **ánodo** del LED amarillo al **pin 12** de Arduino.
- El **cátodo** del LED amarillo va conectado a una **resistencia de 220Ω**, y la resistencia se conecta a **GND**.

3. LED Verde:

- Conecta el **ánodo** del LED verde al **pin 11** de Arduino.
- El **cátodo** del LED verde se conecta a una **resistencia de 220Ω**, y la resistencia se conecta a **GND**.

Paso a Paso en Tinkercad:

1. Abrir Tinkercad:

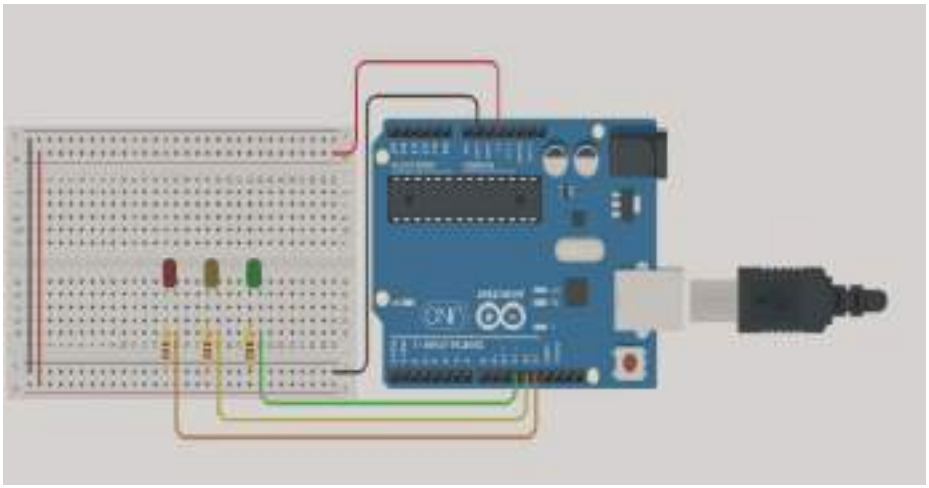
- Inicia sesión en Tinkercad.
- Selecciona **“Circuitos”** y luego haz clic en **“Crear un nuevo circuito”**.

2. Añadir los Componentes:

- Arrastra un **Arduino UNO** al área de trabajo.
- Añade una **protoboard** y los tres **LEDs** (rojo, amarillo y verde).
- Añade tres **resistencias de 220Ω**.

3. Realiza las Conexiones:

- Sigue las conexiones indicadas en el esquema anterior, asegurándote de conectar correctamente los pines de cada LED al Arduino.



Código para Simular el Semáforo:

Una vez que el circuito esté listo, escribe el siguiente código en el editor de Tinkercad.

1. Abrir el Editor de Código:

- Cambia de **"Bloques"** a **"Texto"** para escribir el código en **C++**.

2. Escribir el Código:

```
// Definir pines de los LEDs
int ledRojo = 13;
int ledAmarillo = 12;
int ledVerde = 11;
void setup()
    {
    // Configurar los pines como salidas
    pinMode(ledRojo, OUTPUT);
    pinMode(ledAmarillo, OUTPUT);
    pinMode(ledVerde, OUTPUT);
    }
    void loop()
    {
    // Encender LED verde (paso)
    digitalWrite(ledVerde, HIGH);
    delay(5000); // Mantener 5 segundos
    digitalWrite(ledVerde, LOW);
    // Encender LED amarillo (precaución)
    digitalWrite(ledAmarillo, HIGH);
    delay(2000); // Mantener 2 segundos
    digitalWrite(ledAmarillo, LOW);
```

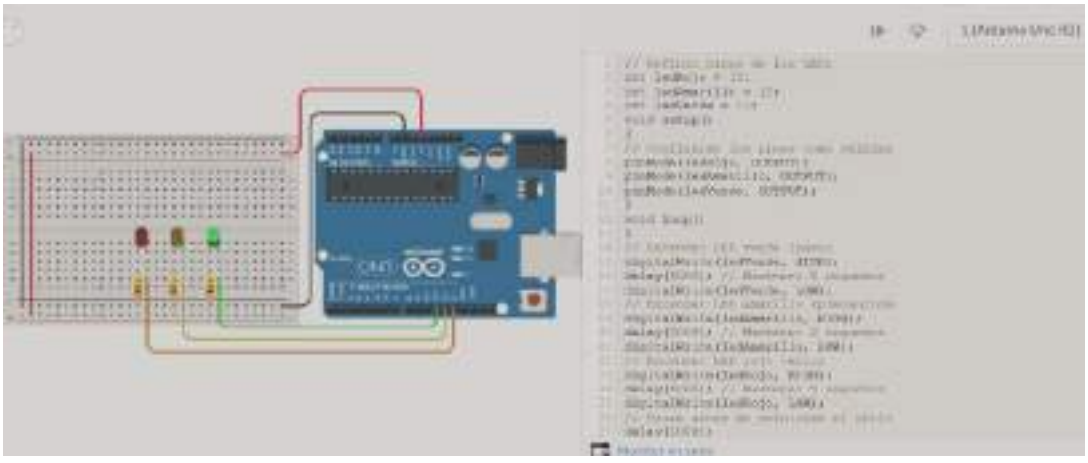
```
// Encender LED rojo (alto)
digitalWrite(ledRojo, HIGH);
delay(5000); // Mantener 5 segundos
digitalWrite(ledRojo, LOW);
// Pausa antes de reiniciar el ciclo
delay(1000);
}
```

Explicación del Código:

- **pinMode(pin, OUTPUT);**: Configura los pines 13, 12 y 11 (donde están conectados los LEDs rojo, amarillo y verde) como salidas.
- **digitalWrite(pin, HIGH);**: Enciende el LED conectado al pin.
- **delay(ms);**: Pausa la ejecución del código por el número de milisegundos especificado.
- El ciclo se repite para simular el comportamiento de un semáforo:
    - **LED verde** encendido por 5 segundos.
    - **LED amarillo** encendido por 2 segundos.
    - **LED rojo** encendido por 5 segundos.

Simulación en Tinkercad:

1. Haz clic en **"Iniciar Simulación"** en la parte superior.
2. Observa cómo el semáforo cambia de verde a amarillo y luego a rojo, siguiendo el ciclo programado.



Conclusión de la práctica:

Este ejercicio simula el comportamiento de un semáforo básico usando tres LEDs y un Arduino. Puedes ajustar los tiempos de cada luz o añadir más funcionalidades, como la simulación de un botón de peatón.

PRÁCTICA

SISTEMA DE RIEGO AUTOMATIZADO CON ARDUINO

Objetivos:

- Simular un sistema de riego automatizado controlado por un **sensor de humedad** del suelo.
- Activar una **bomba de agua** (o un LED como indicador) cuando la humedad del suelo sea baja.

Materiales Necesarios:

- **1 x Arduino UNO**
- **1 x Sensor de Humedad del Suelo**
- **1 x Transistor NPN (2N2222 o similar)**
- **1 x Bomba de agua (puedes simularla con un LED)**
- **1 x Diodo (1N4007)**
- **1 x Resistencia de 220Ω**
- **1 x Fuente de 9V (o una fuente externa para la bomba)**
- **1 x Protoboard**
- **Cables de conexión**

Esquema de Conexión:

1. Conexión del Sensor de Humedad:

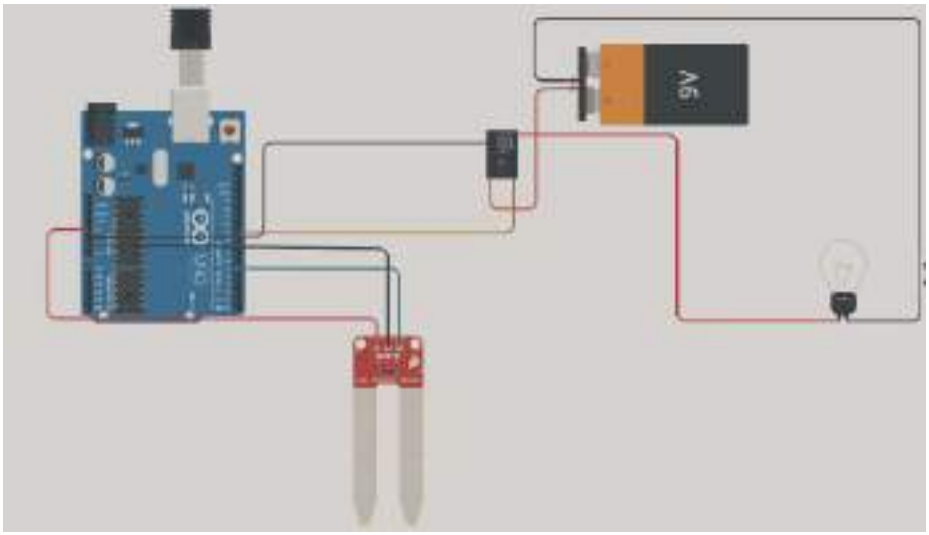
- El sensor de humedad tiene tres pines: **VCC**, **GND** y **OUT**.
- Conecta **VCC** a los 5V del Arduino.
- Conecta **GND** a la tierra del Arduino.
- Conecta el pin **OUT** a una entrada analógica del Arduino, por ejemplo, **A0**.

2. Conexión de la Bomba de Agua (o LED) con el Transistor:

- Conecta el **colector** del **transistor NPN** al negativo de la **bomba de agua** (o un LED si estás simulando).
- Conecta el positivo de la bomba al pin **positivo de la fuente externa** (9V).
- Conecta el **emisor** del transistor a **GND** del Arduino.
- Usa una **resistencia de 220Ω** para conectar la base del transistor al pin digital **D9** del Arduino.
- Coloca un **diodo** entre los terminales de la bomba de agua para proteger el circuito de las corrientes inversas.

Paso a Paso en Tinkercad:

1. **Abrir Tinkercad:**
    - Ve a Tinkercad y selecciona **“Circuitos”** para crear uno nuevo.
2. **Añadir los Componentes:**
    - Arrastra un **Arduino UNO**, un **sensor de humedad**, un **transistor NPN**, una **bomba de agua** (simulada con un LED), y una **fuente de 9V**.
    - Añade también una **resistencia de 220Ω** y un **diodo**.
3. **Realiza las Conexiones:**
    - Conecta el sensor de humedad y el transistor como se indica en el esquema anterior.



Código para el Sistema de Riego Automatizado:

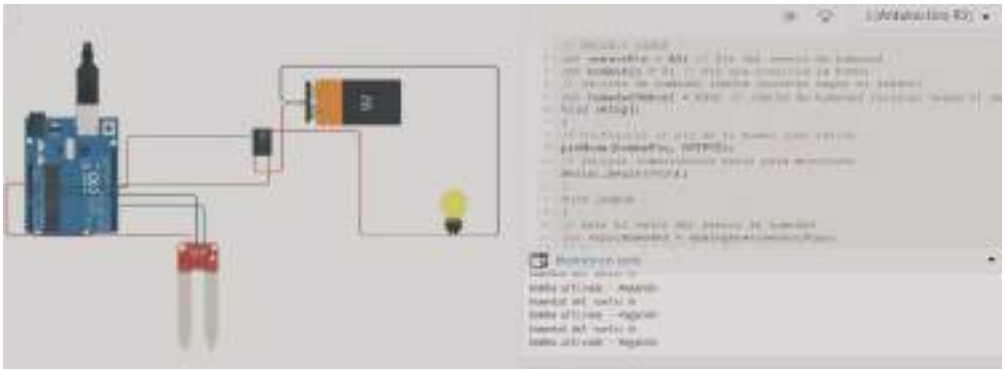
1. **Abrir el Editor de Código:**
    - Cambia el modo de **“Bloques”** a **“Texto”** para escribir el código en **C++**.

2. Escribir el Código:

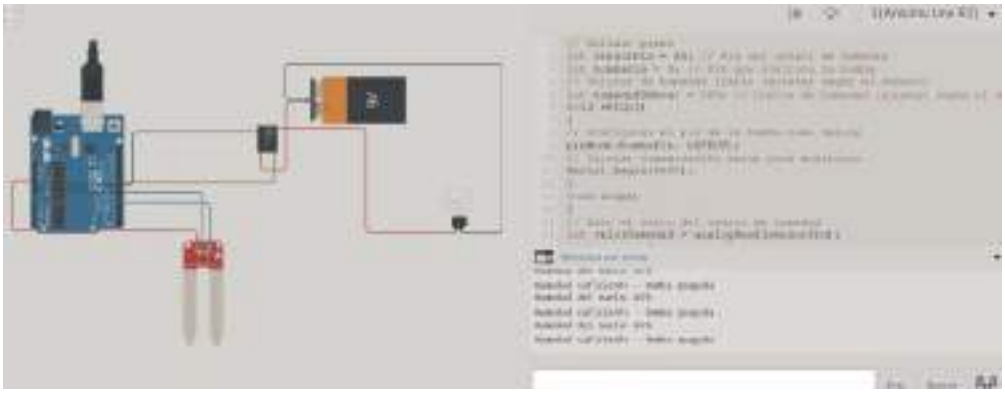
```
// Definir pines
int sensorPin = A0; // Pin del sensor de humedad
int bombaPin = 9; // Pin que controla la bomba
// Valores de humedad límite (ajustar según el sensor)
int humedadUmbral = 600; // Límite de humedad (ajustar según el sensor)
void setup()
        {
        // Configurar el pin de la bomba como salida
        pinMode(bombaPin, OUTPUT);
        // Iniciar comunicación serie para monitoreo
        Serial.begin(9600);
        }
void loop()
        {
        // Leer el valor del sensor de humedad
        int valorHumedad = analogRead(sensorPin);
        // Mostrar el valor en el monitor serie
        Serial.print("Humedad del suelo: ");
        Serial.println(valorHumedad);
        // Comprobar si la humedad está por debajo del umbral
        if (valorHumedad < humedadUmbral)
                {
                // Activar la bomba de agua
                digitalWrite(bombaPin, HIGH);
                Serial.println("Bomba activada - Regando");
                }
                else
                {
                // Apagar la bomba de agua
                digitalWrite(bombaPin, LOW);
                Serial.println("Humedad suficiente - Bomba apagada");
                }
                // Esperar un segundo antes de volver a comprobar
                delay(1000);
        }
```

Explicación del Código:

- **analogRead(sensorPin);**: Lee el valor analógico del sensor de humedad del suelo. El valor será mayor cuando el suelo esté húmedo y menor cuando esté seco.
- **if (valorHumedad < humedadUmbral)**: Si el valor leído es menor que el umbral definido (indicando suelo seco), activa la bomba.



Si no, la apaga.



- **digitalWrite(bombaPin, HIGH/LOW);**: Controla la activación o desactivación de la bomba de agua.

Simulación en Tinkercad:

1. Haz clic en **"Iniciar Simulación"**.
2. Puedes ajustar el valor del sensor de humedad para ver cómo el sistema activa o desactiva la bomba de agua (o el LED si lo estás simulando).

Ajustes del Sistema:

- Si la lectura del sensor es muy variable o no responde como se espera, ajusta el valor de **humedadUmbral**. Esto dependerá del tipo de sensor de humedad que uses.
- Puedes cambiar el **pin de control de la bomba** si tienes una configuración diferente de hardware.

Conclusión de la práctica

Este sistema de riego automatizado utiliza un **sensor de humedad** para medir la cantidad de agua en el suelo. Cuando el suelo está seco, la bomba se activa para regar. Este proyecto es ideal para aplicaciones de **jardinería automatizada** y puede ser mejorado con funcionalidades adicionales como **monitoreo remoto** o **control por tiempo**.

Bibliografía

Dorf, R. C., & Svoboda, J. A. (2018). *Introduction to Electric Circuits.* Wiley.

Floyd, T. L. (2018). *Electronic Devices.* Pearson Education.

Graf, R. F. (1999). *Modern dictionary of electronics.* Butterworth-Heinemann: Newnes.

Malvino, A., & Bates, D. J. (2007). *Principios de Electrónica.* Madrid: McGraw-Hill.

Monk, S. (2017). *Electronic Cookbook, Practical Electronic Recipes with Arduino & Raspberry PI.* O'Reilly Media, Inc.

UNIDAD 03

Bloques Constructivos para Circuitos Combinacionales

Bloques Constructivos para Circuitos Combinacionales

TEMA 3.1: INTRODUCCIÓN A LA CONSTRUCCIÓN DE CIRCUITOS DIGITALES COMBINACIONALES DE MEDIANA COMPLEJIDAD

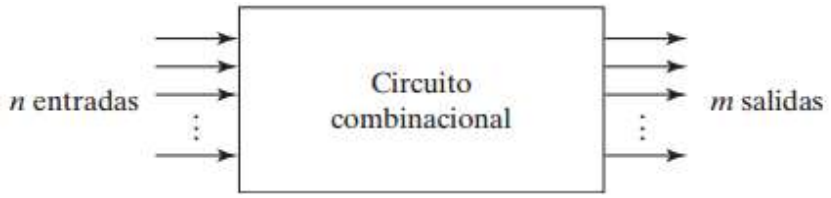
Los circuitos lógicos para sistemas digitales pueden ser combinacionales o secuenciales. Un circuito **combinacional** consiste en compuertas lógicas cuyas salidas en cualquier momento están determinadas por la combinación actual de entradas. Los circuitos **secuenciales** usan elementos de almacenamiento además de compuertas lógicas, y sus salidas son función de las entradas y del estado de los elementos de almacenamiento. En esta unidad revisaremos como se construyen estos circuitos, y sus aplicaciones principales.

3.1.1 Circuitos combinacionales

Un circuito combinacional realiza una operación que se puede especificar lógicamente con un conjunto de funciones booleanas. Implementan una o varias funciones de conmutación, tales que las salidas del circuito en cada instante de tiempo dependen única y exclusivamente de las **señales de entrada en el mismo instante** (Deschamps, Valderrama, & Terés, 2017).

Un circuito combinacional consiste en **variables de entrada**, **compuertas lógicas** y **variables de salida**. Las compuertas lógicas aceptan señales de las entradas y generan señales para las salidas. Este proceso transforma información binaria, de los datos de entrada dados a los datos de salida requeridos.

En la figura a continuación se presenta un diagrama de bloques de un circuito combinacional. Las ***n*** variables binarias de entrada provienen de una fuente externa; las ***m*** variables de salida van a un destino externo. Cada variable de entrada y de salida existe físicamente como una señal binaria que representa 1 lógico y 0 lógico.



Principales palabras reservadas de VHDL

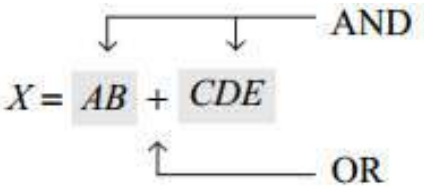
Hay varios circuitos combinacionales que se usan ampliamente en el diseño de sistemas digitales. Esos circuitos pueden conseguirse en circuitos integrados y se clasifican como componentes estándar. En esta unidad presentaremos los circuitos combinacionales estándar más importantes, como los sumadores, restadores, comparadores, decodificadores, codificadores y multiplexores. (Morris Mano, 2003).

3.1.2 Análisis y diseño de circuitos combinacionales

El análisis de un circuito combinacional requiere deducir la función que realiza el circuito. Para este proceso se parte de un diagrama lógico dado y culmina en un conjunto de **funciones booleanas**, una **tabla de verdad** o una posible **explicación** del funcionamiento del circuito. Si ya se tienen estos elementos (tabla de verdad o función booleana), el análisis es más sencillo. El análisis se efectúa manualmente, o bien, utilizando un programa de simulación en computadora.

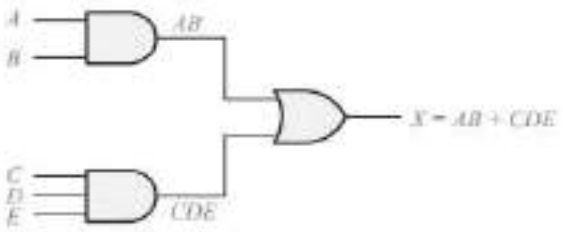
El primer paso del análisis consiste en asegurarse de que el circuito dado sea combinacional y no secuencial. El diagrama de un circuito combinacional tiene compuertas lógicas sin trayectorias de retroalimentación ni elementos de memoria (Morris Mano, 2003).

Cuando se tiene una función, la generación del circuito puede ser fácilmente obtenida. Consideremos la expresión booleana $X = AB + CDEX = AB + CDE$. Una rápida inspección revela que esta expresión está formada por dos términos, *AB* y *CDE*, y tienen un dominio de cinco variables. El primer término está definido por la operación AND entre A y B, y el segundo término queda definido por la multiplicación (AND) de C, Dy E. Después, los dos términos se suman (operación OR) para dar lugar a la salida X. Observe que, en esta expresión, las operaciones AND son dos términos individuales, AB y CDE, que deben efectuarse antes de aplicar la operación OR.



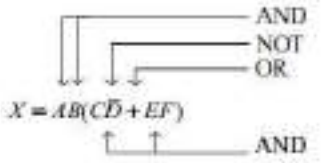
Análisis de la función *X=AB+CDE*

Para implementar esta expresión booleana se requiere una puerta AND de 2 entradas para obtener el término AB, una puerta AND de tres entradas para generar el término CDE, y una puerta OR de 2 entradas para combinar los dos términos obtenidos en las puertas AND. El resultado se muestra en la Figura a continuación: (Floyd, 2006).

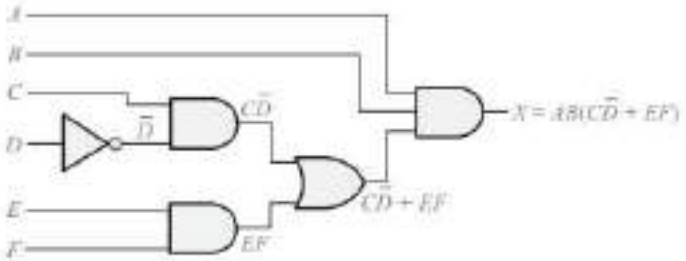


Circuito lógico para *X=AB+CDE*

Veamos otro ejemplo, vamos a implementar la siguiente expresión $X = AB(C\overline{D} + EF)X = AB(C\overline{D} + EF)$. Un examen de esta expresión muestra que se aplica la operación AND a los términos *AB* y $(C\overline{D} + EF)(C\overline{D} + EF)$. El término $C\overline{D} + EFC\overline{D} + EF$ se obtiene aplicando primero la operación AND a *C* y $\overline{D}\overline{D}$ y luego a *E* y *F*, y aplicando por último la operación OR a estos dos términos. Previamente, se debe aplicar el operador NOT a *D*. Las operaciones lógicas deben efectuarse en el orden adecuado. A continuación el análisis de la función y el circuito resultante:



Análisis de $X = AB(C\overline{D} + EF)$



Circuito lógico para $X = AB(C\overline{D} + EF)$

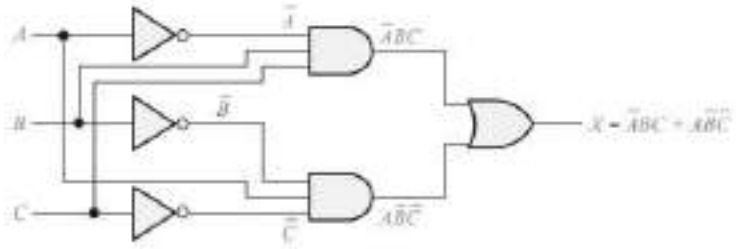
Es posible que la función se pueda simplificar o transformar para reducir el número de compuertas lógicas. Se puede hacer un análisis al respecto, pero por ahora nos quedaremos con el análisis simple.

De forma similar, si en lugar de partir de una expresión se parte de una tabla de verdad, puede escribirse la suma de productos que se obtiene de la tabla de verdad, y luego implementar el circuito lógico. Consideremos la tabla de verdad a continuación.

Entradas			Salida	
A	*B*	*C*	*X*	Término producto
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\overline{A}BC$
1	0	0	1	$A\overline{B}\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	0	

Tabla de verdad

La expresión booleana suma de productos que se obtiene a partir de la tabla de verdad, haciendo la suma (OR) de los productos para los que X= 1 es $X = \bar{A}BC + A\bar{B}\bar{C}X = \bar{A}BC + A\bar{B}\bar{C}$. Las puertas lógicas necesarias para implementar esta expresión son: tres inversores para obtener las variables negadas, dos puertas AND de 3 entradas para obtener los dos términos productos, y una puerta OR de 2 entradas. A continuación se ve el circuito obtenido a partir de la tabla de verdad.



Circuito lógico para $X = \bar{A}BC + A\bar{B}\bar{C}X = \bar{A}BC + A\bar{B}\bar{C}$

En este caso primero se obtuvo la función booleana correspondiente para la tabla de verdad, como una suma de términos producto, y luego se obtuvo el circuito.

Tema 3.2: Circuitos aritméticos: sumadores, restadores, comparadores.

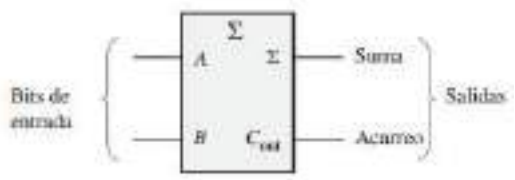
Los circuitos aritméticos son relativamente sencillos de elaborar. Como veremos a continuación, se puede generar un circuito aritmético básico utilizando las compuertas que hemos visto.

3.2.1 Sumadores básicos

Los sumadores son muy importantes en muchos tipos de sistemas digitales en los que se procesan datos numéricos. Si recordamos las reglas de la suma binaria para dos bits, tendremos el contenido a continuación a la izquierda. Estas operaciones se pueden realizar mediante un circuito lógico denominado **semisumador**. Un semisumador admite dos dígitos binarios en sus entradas y genera dos dígitos binarios en sus salidas: un bit de suma y un bit de acarreo. (Floyd, 2006).

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

Suma binaria de dos bits



Símbolo lógico de un semi-sumador

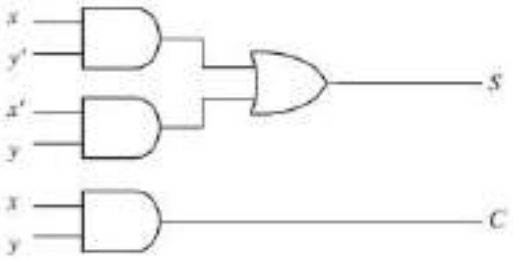
Por la descripción verbal del semisumador, se sabe que este circuito necesita dos entradas binarias y dos salidas binarias. Las variables de entrada designan los bits sumandos; las de salida, la suma y el acarreo. Asignaremos los símbolos ***x*** y

y a las dos entradas, y *S* (de suma) y *C* (de *carry*) a las salidas. A continuación se muestra la tabla de verdad correspondiente:

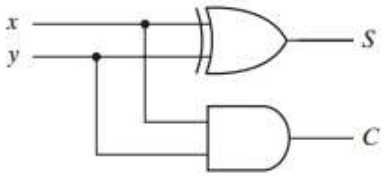
x	*y*	*C*	*S*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabla de verdad de un semisumador

Las funciones booleanas para las dos salidas se obtienen directamente de la tabla de verdad. Las expresiones simplificadas en suma de productos son $S = \bar{x}y + x\bar{y}S = \bar{x}y + x\bar{y}$, y $C = xyC = xy$ (Morris Mano, 2003). En la figura a continuación a la izquierda, se puede ver la implementación del semisumador con compuertas NOT, AND y OR. También se puede implementar con un OR exclusivo (XOR) y una compuerta AND, como se ve en la figura de la derecha.

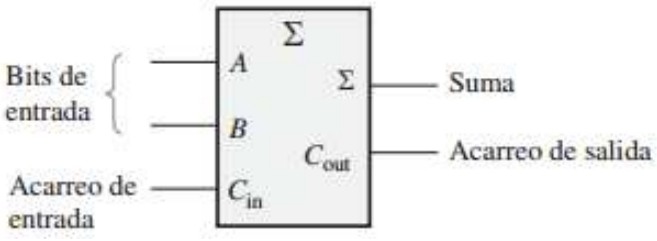


Semisumador con compuertas básicas



Semisumador con XOR y AND

El segundo tipo de sumador es el **sumador completo**. Un sumador acepta dos bits de entrada, un **acarreo de entrada**, y genera una salida de suma y un acarreo de salida. La diferencia principal entre un sumador completo y un semisumador es que el sumador completo acepta un acarreo de entrada (Floyd, 2006).



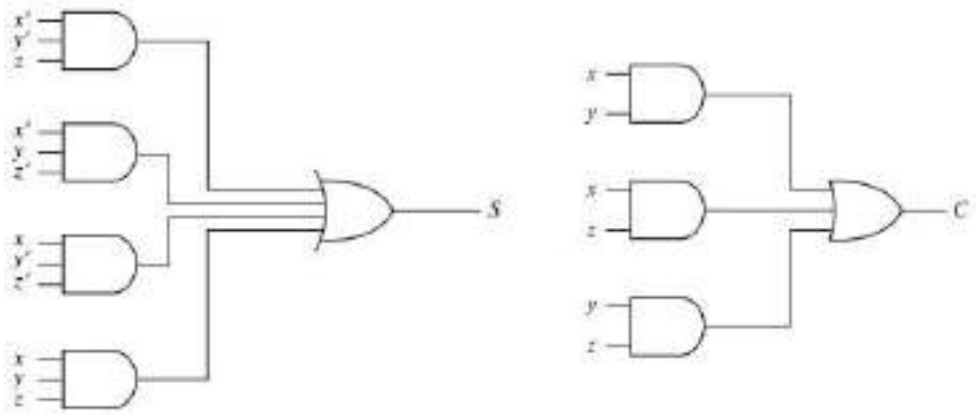
Símbolo lógico de un sumador completo

Las dos salidas se designan otra vez con los símbolos *S* y *C*. La variable binaria *S* da el valor del bit menos significativo de la suma. La variable binaria *C* da el acarreo de salida. La tabla de verdad del sumador completo se presenta en la tabla a continuación

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

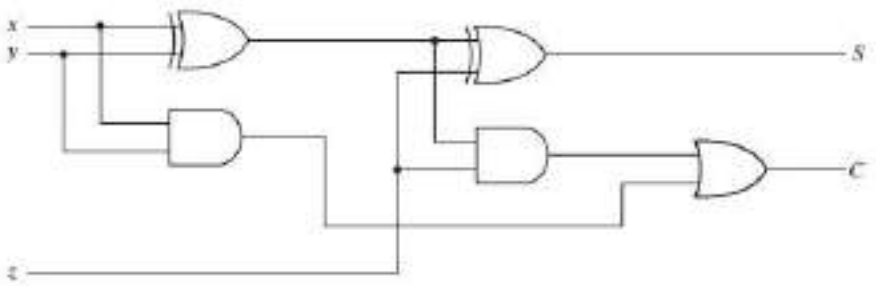
Tabla de verdad de un sumador completo

Una vez obtenidos las expresiones simplificadas para cada una de las salidas, el resultado es $S = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyzS = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$, y $C = xy + xz + yzC = xy + xz + yz$.



Implementación de un sumador completo como suma de productos

En el caso del resultado de la suma, también puede expresarse con OR exclusivos, quedando $S = z \oplus (x \oplus y)S = z \oplus (x \oplus y)$. De esta forma, el sumador completo también puede implementarse con dos semisumadores y una compuerta OR (Morris Mano, 2003). La figura a continuación muestra dicha implementación:



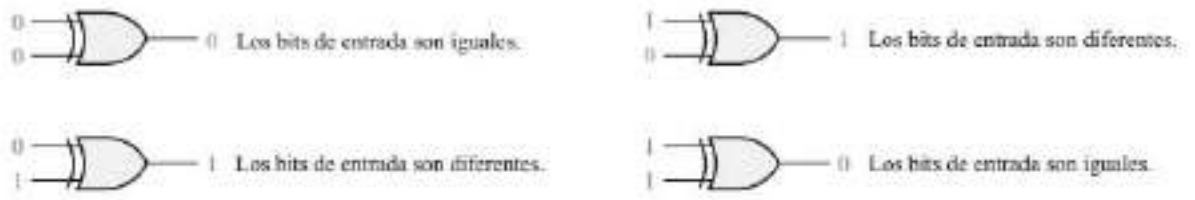
Implementación de un sumador completo con dos semisumadores y una compuerta OR

3.2.2 Restadores

La forma más conveniente de efectuar la resta de números binarios sin signo es utilizando complementos. La resta *A-B* se efectúa obteniendo el **complemento a dos** de *B* y sumándolo a *A*. El complemento a uno se implementa con inversores, y el 1 se suma a través del acarreo de entrada. El circuito para restar *A-B* consiste en un sumador con inversores colocados entre cada entrada de datos *B* y la entrada correspondiente del sumador completo. El acarreo de entrada C_0 debe ser igual a 1 al restar. La operación se convierte entonces en *A* más el complemento a uno de *B* más 1, es decir, igual a *A* más el complemento a dos de *B* (Morris Mano, 2003).

3.2.3 Comparadores

La función básica de un comparador consiste en **comparar** las magnitudes de dos cantidades binarias para determinar su relación. En su forma más sencilla, un circuito comparador determina si dos números son iguales. La puerta XOR se puede emplear como un comparador básico, ya que su salida es 1 si sus dos bits de entrada son diferentes y 0 si son iguales.



Funcionamiento del comparador básico

Para comparar números binarios de dos bits, se necesita una puerta XOR adicional. Los dos bits menos significativos (*LSB*) de ambos números se comparan mediante la puerta G_1 y los dos más significativos (*MSB*) son comparados mediante la puerta G_2, como se muestra en la figura a continuación. Si los dos números son iguales, sus correspondientes bits también lo son, y la salida de cada puerta XOR será 0. Si los bits no son idénticos, la salida de la puerta XOR será un 1 (Floyd, 2006).

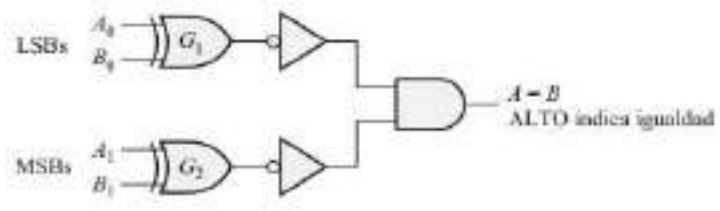
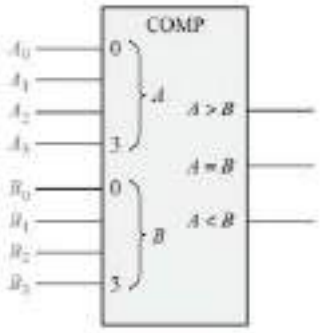


Diagrama lógico de la comparación de igualdad de dos números de 2 bits

Además de una salida que indica igualdad, muchos circuitos comparadores tienen salidas adicionales que indican cuál de los dos números que se comparan es el mayor. Esto significa que existe una salida que indica cuándo el número *A* es

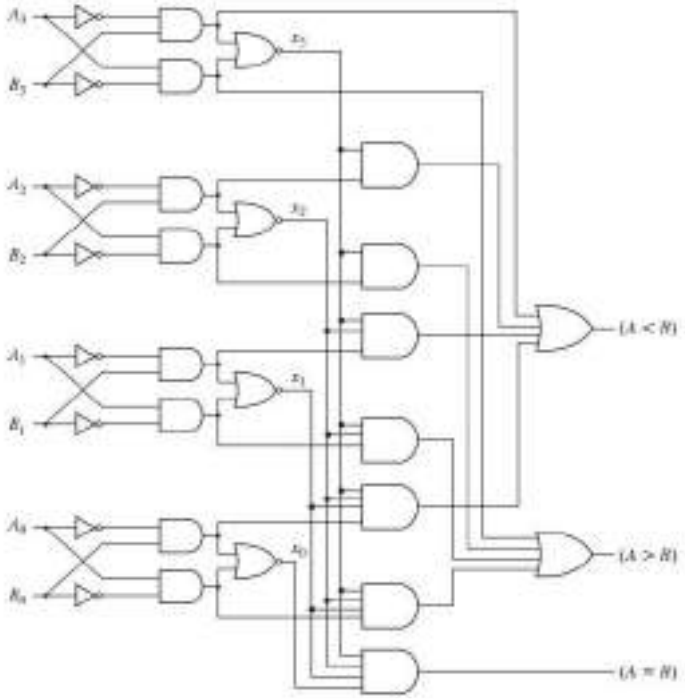
mayor que el número *B* (*A*> *B*) y otra salida que indica cuándo *A* es menor que *B* (*A*< *B*) (Floyd, 2006).

Para determinar si A es mayor o menor que B, se inspeccionan las magnitudes relativas de pares de dígitos significativos, comenzando por la posición más significativa. Si los dos dígitos son iguales, se comparará el siguiente par de dígitos menos significativos.



Símbolo lógico para un comparador de 4 bits con indicación de desigualdad

La comparación sucesiva se expresa lógicamente con las dos funciones booleanas, una para *A*>*B* y otra para *A*<*B*. La implementación con compuertas de las tres variables de salida es más sencilla de lo que parece porque implica cierta repetición. Las salidas de desigualdad utilizan las mismas compuertas que generan la salida de igualdad (Morris Mano, 2003). El diagrama lógico del comparador de cuatro bits es este:



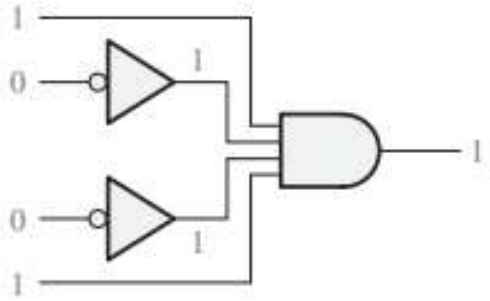
Comparador de magnitudes de cuatro bits

Tema 3.3: Codificadores, decodificadores y multiplexores.

3.3.1 Decodificadores

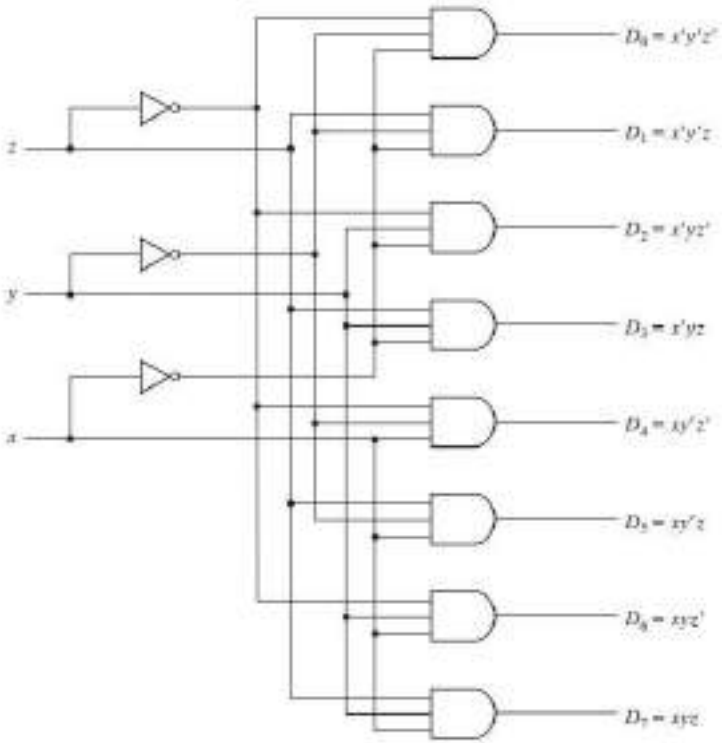
En los sistemas digitales, las cantidades discretas de información se representan con códigos binarios. Un código binario de ***n*** bits puede representar hasta 2^n elementos distintos de información codificada. Un **decodificador** es un circuito combinacional que convierte información binaria de *n* líneas de entrada a un máximo de 2^n líneas de salida distintas. Si la información codificada en *n* bits tiene combinaciones que no se usan, el decodificador podría tener menos de 2^n salidas (Morris Mano, 2003).

Veamos un ejemplo: Supongamos que necesitamos determinar cuándo aparece el número binario 1001 en las entradas de un circuito digital. Se puede utilizar una puerta AND como elemento básico de decodificación, ya que produce una salida a nivel ALTO sólo cuando todas sus entradas están a nivel ALTO (Floyd, 2006). Por tanto, debe asegurarse de que todas las entradas de la puerta AND estén a nivel ALTO cuando se introduce el número 1001, lo cual se puede conseguir invirtiendo los dos bits centrales (cuyos bits son 0), como se muestra en la figura siguiente:



Lógica de decodificación del código binario 1001 con una salida activa a nivel ALTO

Ahora, consideremos el circuito decodificador de 3 a 8 líneas de la figura siguiente. Las **tres entradas** se decodifican para dar **ocho salidas**, cada una de las cuales representa uno de los mintérminos de las tres variables de entrada. Los tres inversores producen el complemento de las entradas, y cada una de las ocho compuertas AND genera uno de los mintérminos. Una aplicación específica de este decodificador es la conversión de **binario a octal**. Las variables de entrada representan un número binario, y las salidas, los ocho dígitos del sistema numérico octal. Sin embargo, un decodificador de 3 a 8 líneas puede servir para decodificar cualquier código de tres bits y obtener ocho salidas, una por cada elemento del código.



Decodificador de 3 a 8 líneas

El funcionamiento del decodificador podría aclararse al examinar la tabla de verdad mostrada a continuación. Para cada posible combinación de entrada, hay siete salidas que son 0 y sólo una igual a 1. La salida que vale 1 representa el mintérmino equivalente al número binario que se está alimentando a las líneas de entrada (Morris Mano, 2003).

Entradas			Salidas							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

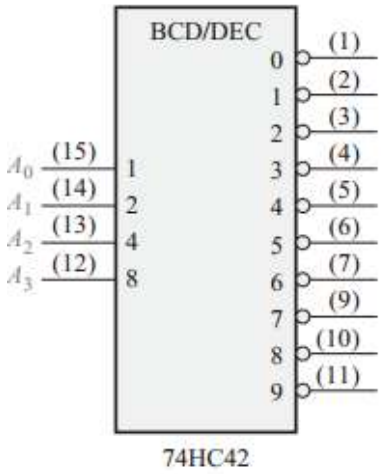
Tabla de verdad de un decodificador de 3 a 8 líneas

Algunos decodificadores se construyen con compuertas NAND. Puesto que una compuerta NAND produce la operación AND con la salida invertida, resulta más económico generar los mintérminos del decodificador en su forma complementada.

Los decodificadores se utilizan en muchos tipos de aplicaciones. Un ejemplo es la selección de entradas y salidas en las computadoras. Veamos un par de ejemplos de decodificadores específicos

El decodificador BCD a decimal:

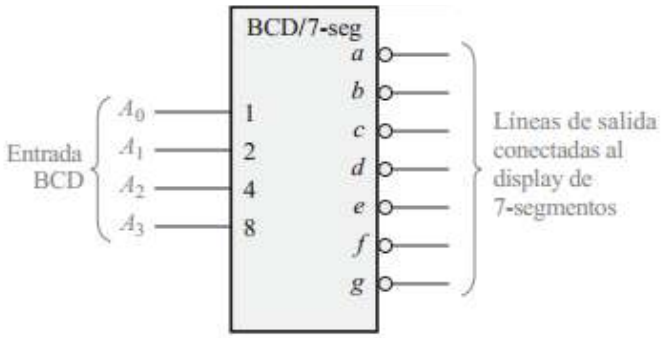
Un decodificador BCD a decimal convierte cada código BCD (código 8421) en uno de los diez posibles dígitos decimales. Frecuentemente, se le **denomina decodificador de 4-líneas a 10-líneas** o decodificador 1 de 10. Este decodificador tiene 4 líneas de entrada, pero solo 10 líneas de salida, no $2^4=16$ como se podría pensar, dado que el código BCD sólo representa los diez dígitos decimales de 0 a 9. La siguiente figura muestra las señales de entrada y de salida de un chip T4HC42, que es un decodificador BCD-decimal.



El decodificador BCD a decimal 74HC42

El decodificador BCD a 7-segmentos:

El decodificador BCD a 7-segmentos acepta el código BCD en sus entradas y proporciona salidas capaces de excitar un **display de 7-segmentos** para generar un dígito decimal. En la figura a continuación se muestra el diagrama lógico de un decodificador básico de 7-segmentos (Floyd, 2006).



El decodificador BCD a decimal 74HC42

3.3.2 Codificadores

Un **codificador** es un circuito lógico combinacional que, esencialmente, realiza la función "inversa" del decodificador. Un codificador permite que se introduzca en una de sus entradas un nivel activo que representa un dígito, como puede ser un dígito decimal u octal, y lo convierte en una salida codificada, como BCD o binario. Los codificadores se pueden diseñar también para codificar símbolos diversos y caracteres alfabéticos. El proceso de conversión de símbolos comunes o números a un formato codificado recibe el nombre de **codificación** (Floyd, 2006).

El codificador tiene 2^n (o menos) líneas de entrada y ***n*** líneas de salida. Estas últimas generan el código binario correspondiente al valor de entrada. Un ejemplo de codificador es el codificador de octal a binario cuya tabla de verdad se presenta en la tabla siguiente. Tiene ocho entradas (una para cada uno de los dígitos octales) y tres salidas que generan el número binario correspondiente. Se supone que **sólo una entrada** es igual a 1 en cualquier momento dado. El codificador se puede implementar con compuertas OR cuyas salidas se determinan directamente de la tabla de verdad.

Entradas								**Salidas**		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

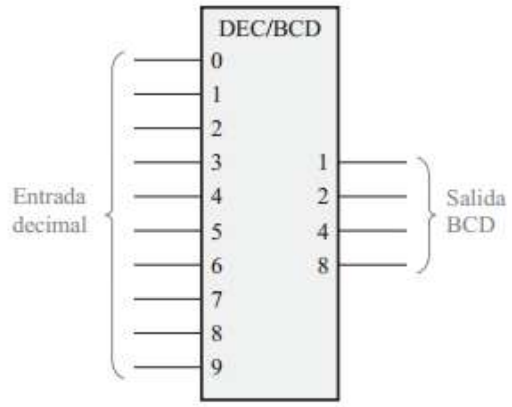
Tabla de verdad del codificador de octal a binario

La salida ***z*** es 1 cuando el dígito octal de entrada es 1, 3, 5 o 7. La salida ***y*** es 1 para los dígitos octales 2, 3, 6 o 7, y la salida ***x*** es 1 para los dígitos 4, 5, 6 o 7. Estas condiciones se expresan con las funciones booleanas de salida siguientes:

$$z=D_1+D_3+D_5+D_7$$
$$y=D_2+D_3+D_6+D_7$$
$$x=D_4+D_5+D_6+D_7$$ (Morris Mano, 2003)

Codificador decimal-BCD

Este tipo de codificador tiene diez entradas, una para cada dígito decimal, y cuatro salidas que corresponden al código BCD, como se muestra en la figura siguiente. Este es un codificador básico de 10-líneas a 4-líneas.



Símbolo lógico de un codificador decimal a BCD

A partir de esta tabla de verdad del código BCD (que vimos anteriormente) podemos determinar la relación entre cada bit BCD y los dígitos decimales, con el fin de analizar la lógica. Por ejemplo, el bit más significativo del código BCD, A_3, es siempre un 1 para los dígitos decimales 8 o 9. La expresión OR para el bit A_3 en función de los dígitos decimales puede por tanto escribirse como $A_3=8 + 9$; y así para cada salida A_2, A_1 y A_0 del codificador. La siguiente figura muestra cómo se puede implementar el codificador utilizando compuertas OR a los dígitos decimales de entrada apropiados, para así formar cada salida BCD. No se necesita una entrada para el dígito 0, ya que las salidas BCD están todas a nivel BAJO cuando no hay entradas a nivel ALTO (Floyd, 2006).

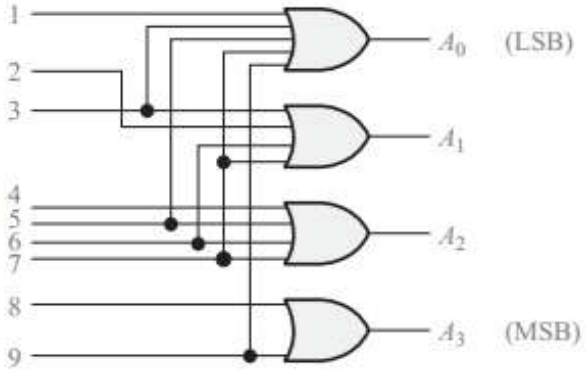


Diagrama lógico básico de un codificador decimal a BCD

3.3.3 Multiplexores

Un **multiplexor** (*MUX*) es un dispositivo que permite dirigir la información digital procedente de **diversas fuentes** a **una única línea** para ser transmitida a través de dicha línea. El multiplexor básico posee varias líneas de entrada de datos y una única línea de salida. También posee entradas de selección de datos, que permiten conmutar los datos digitales provenientes de cualquier entrada hacia la línea de salida. A los multiplexores también se les conoce como **selectores de datos** (Floyd, 2006).

La selección de una línea de entrada dada se controla con un conjunto de líneas de selección. Normalmente, hay 2^n líneas de entrada y ***n*** líneas de selección cuyas combinaciones de bits determinan cuál entrada se selecciona. Un multiplexor de 2 líneas a 1 conecta una de dos fuentes de un bit a un destino común, como se indica en las figuras siguientes. El circuito tiene dos líneas de entrada de datos, una línea de salida y una línea de selección *S*. Cuando *S=0*, se habilita la compuerta AND de arriba e I_0 cuenta con una trayectoria hacia la salida. Cuando *S=1*, la compuerta AND inferior está habilitada e I_1 tiene una trayectoria hacia la salida. El multiplexor actúa como un interruptor electrónico que selecciona una de dos fuentes. El diagrama de bloques de un multiplexor a veces se representa con un símbolo en forma de **cuña**, como en la figura a la derecha. Esto sugiere visualmente cómo una fuente de datos, seleccionada de entre varias, se dirige a un solo destino (Morris Mano, 2003).

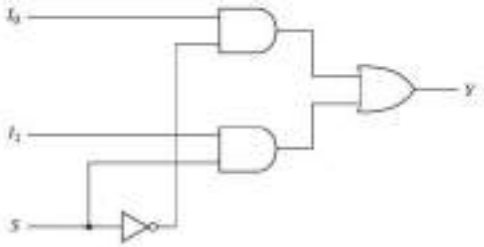


Diagrama lógico de un Multiplexor

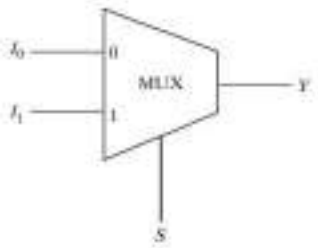


Diagrama de bloque de un Multiplexor

A continuación, se presenta un multiplexor de 4 líneas a 1. Cada una de las cuatro entradas, I_0 a I_3, se aplica a una entrada de una compuerta AND. Las líneas de selección S_1 y S_0 se decodifican para seleccionar una compuerta AND determinada. Las salidas de las compuertas AND se aplican a una sola compuerta OR que genera la salida de una sola línea. La tabla de la función indica qué entrada se pasa a la salida con cada combinación de los valores binarios de selección.

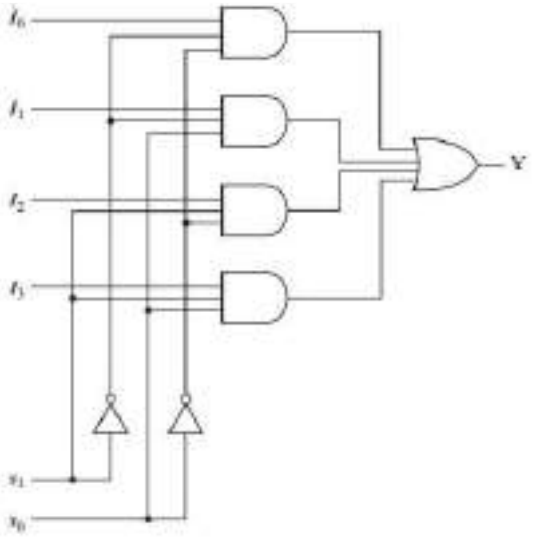


Diagrama lógico de un Multiplexor de 4 líneas a 1

Para ilustrar el funcionamiento del circuito, consideremos el caso en que $S_1S_0=10$. La compuerta AND asociada a la entrada I_2 tiene 1 en dos de sus entradas, y la tercera conectada a I_2. Las otras tres compuertas AND tienen 0 en por lo menos una de sus entradas, lo que hace que produzcan 0 como salida. Así, la salida de la compuerta OR tiene el mismo valor que I_2, así que constituye un camino de la entrada seleccionada hasta la salida. Los multiplexores también se denominan selectores de datos (Morris Mano, 2003).

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Tabla de verdad del Multiplexor de 4 a 1

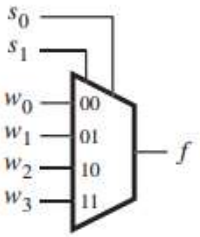


Diagrama de bloque de un Multiplexor de 4 a 1

Tema 3.4: Diseño de circuitos combinacionales de mediana complejidad utilizando bloques constructivos básicos

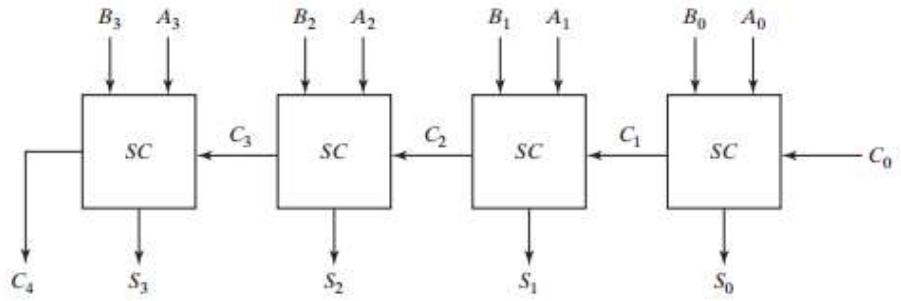
Ya hemos visto circuitos combinacionales básicos. Estos circuitos básicos pueden tomarse como bloques para construir circuitos más complejos. Por ejemplo, se podría tomar varios bloques de sumador completo (que suma 2 bits) para sumar dos cantidades de varios bits cada una. Veremos algunos ejemplos.

4.4.1 Circuitos aritméticos

Antes ya vimos los circuitos básicos para operaciones matemáticas, pero en casi todos los casos fueron para realizar operaciones con 2 bits. Ahora veremos cómo trabajar con más bits.

Sumador binario

Un **sumador binario** es un circuito digital que produce la suma aritmética de dos números binarios. Es posible construirlo con sumadores completos dispuestos en cascada, conectando el **acarreo de salida** de cada sumador completo **al acarreo de entrada** del siguiente sumador completo de la cadena. La figura a continuación muestra la interconexión de cuatro circuitos sumadores completos (***SC***) para formar un sumador binario de cuatro bits con acarreo rizado.



Sumador de cuatro bits

Los bits de los sumandos *A* y *B* se designan con subíndices de izquierda a derecha; el subíndice *0* denota el bit menos significativo. Los acarreos se conectan en una cadena a través de los sumadores completos. El acarreo de entrada del sumador es C_0 y se propaga a través de los sumadores completos hasta el acarreo de salida C_4. Las salidas *S* generan los bits de suma requeridos. Un sumador de *n* bits requiere *n* sumadores completos con cada acarreo de salida conectado al acarreo de entrada del siguiente sumador completo de orden superior.

Los bits se suman con sumadores completos, comenzando por la posición menos significativa (subíndice *0*) para formar el bit de suma y el bit de acarreo. El acarreo de entrada C_0 en la posición menos significativa debe ser 0. El valor de C_{i+1} en una posición significativa dada es el acarreo de salida del sumador completo anterior.

El sumador de cuatro bits es un ejemplo representativo de un componente estándar. Observe que el diseño de este circuito empleando el método clásico requeriría una tabla de verdad con 2^9=512 entradas, ya que el circuito tiene nueve entradas. Al usar un método iterativo de conectar en cascada una función estándar, es posible obtener una implementación sencilla y directa (Morris Mano, 2003).

Así, para hacer un sumador que sume dos números de ***n*** bits, se necesitarían ***n*** sumadores completos de dos bits más acarreo de entrada. También es posible hacer una expansión de sumadores. Por ejemplo, se podría tomar 2 sumadores de 4 bits para sumar números de 8 bits.

Sumador BCD.

Consideremos la suma aritmética de dos dígitos decimales en BCD, junto con un acarreo de entrada de una etapa anterior. Puesto que ninguno de los dígitos de entrada es mayor que 9, la suma de salida no puede ser mayor que 9+9+1=19, donde el 1 de la suma es el acarreo de entrada. Suponga que aplicamos dos dígitos BCD a un sumador binario de cuatro bits. El sumador formará la suma en binario y producirá un resultado entre 0 y 19. El problema es que cuando la suma binaria es mayor que 1001, se obtiene una representación no válida en BCD.

En la figura siguiente se observa un **sumador BCD** que suma dos dígitos BCD y produce un dígito de suma en BCD. Primero se suman los dos dígitos decimales, junto con el acarreo de entrada, en el sumador de cuatro bits de la parte de arriba, para producir la suma binaria. Si el acarreo de salida es 0, no se suma nada a la suma binaria. Si es 1, se suma 0110 binario a la suma binaria con el sumador de cuatro bits de la parte de abajo. El acarreo de salida generado por el sumador de abajo se desecha, pues proporciona información con que ya se cuenta en la terminal de acarreo de salida. Un sumador decimal en paralelo que suma ***n*** dígitos decimales necesita ***n*** etapas de sumador BCD. El acarreo de salida de una etapa deberá conectarse al acarreo de entrada de la siguiente etapa de orden superior.

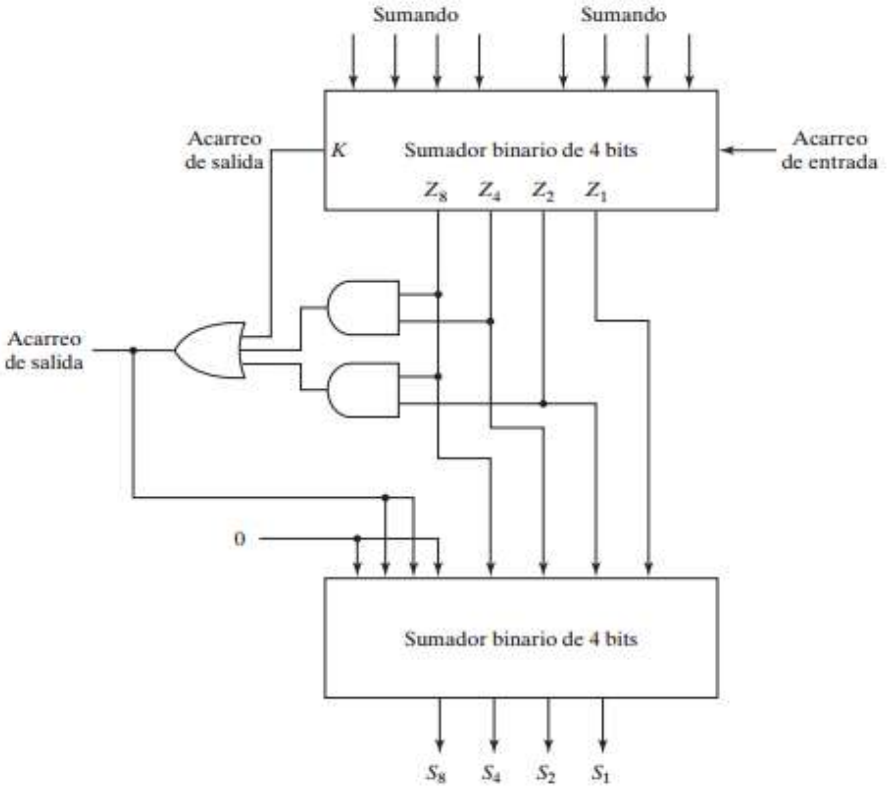
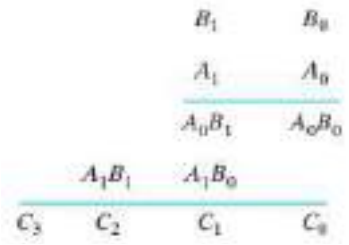


Diagrama de bloques de un sumador BCD

Multiplicador binario.

La multiplicación de números binarios se efectúa igual que la de números decimales. El multiplicando se multiplica por cada bit del multiplicador, comenzando por el bit menos significativo. Cada una de estas multiplicaciones forma un producto parcial. Los productos parciales sucesivos se desplazan una posición a la izquierda. El producto final se obtiene **sumando los productos parciales**.

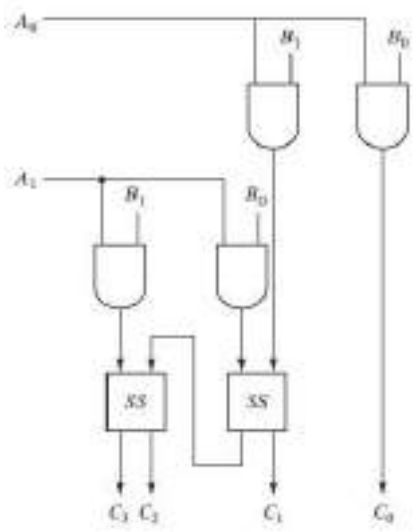


Esquema de la multiplicación de dos números de dos bits

Para ver cómo puede implementarse un multiplicador binario con un circuito combinacional, consideremos la multiplicación de dos números de dos bits, como se muestra en la figura siguiente. Los bits del multiplicando son B_1 y B_0, los bits del multiplicador son A_1 y A_0, y el producto es $C_3C_2C_1C_0$. El primer producto parcial se forma multiplicando A_0 por B_1B_0. La multiplicación de dos bits como A_0 y B_0 produce

1 si ambos bits son 1; de lo contrario, produce 0. Esto es idéntico a la operación AND. Por tanto, el producto parcial puede implementarse con compuertas AND como se indica en el diagrama. El segundo producto parcial se forma multiplicando A_1 por B_1B_0 y se desplaza una posición a la izquierda.

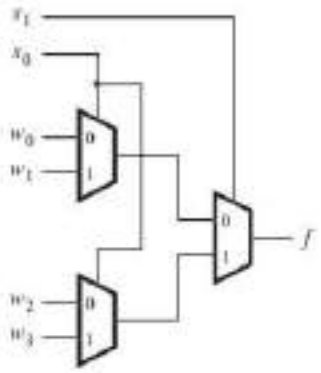
Los dos productos parciales se suman con dos circuitos de semisumador (*SS*). Por lo regular, los productos parciales tienen más bits, y ello obliga a usar sumadores completos para obtener la suma de los productos parciales. El bit menos significativo del producto no tiene que pasar por un sumador porque se forma con la salida de la primera compuerta AND (Morris Mano, 2003).



Esquema de la multiplicación de dos números de dos bits

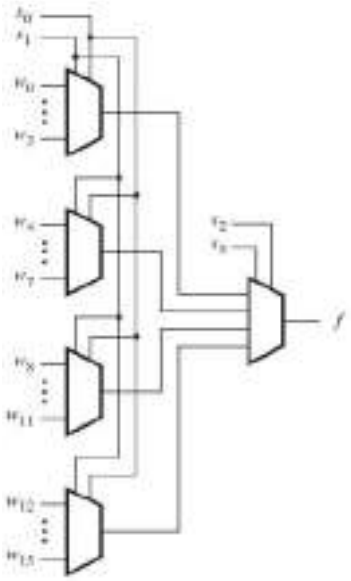
3.4.2 Circuitos con Multiplexores

Los multiplexores son muy útiles. Los multiplexores más grandes también pueden construirse a partir de otros más pequeños. Por ejemplo, el multiplexor cuatro a uno puede construirse con tres multiplexores dos a uno, como se ilustra en la figura siguiente. Si el multiplexor cuatro a uno se implementa con compuertas de transmisión, entonces la estructura de esta fi gura siempre se utiliza.



Uso de multiplexores dos a uno para construir un multiplexor cuatro a uno

Si el multiplexor cuatro a uno se implementa con compuertas de transmisión, entonces la estructura de esta figura siempre se utiliza. En la figura a continuación se muestra cómo se construye un multiplexor 16 a 1 con cinco multiplexores cuatro a uno.



Multiplexor 16 a 1

También se puede interpretar las tablas de verdad para implementar funciones lógicas mediante multiplexores. En cada caso, las entradas a los multiplexores son las constantes 0 y 1, o alguna variable o su complemento. Además de usar tales entradas simples, es posible conectar circuitos más complejos como entradas a un multiplexor, lo que permite que las funciones se sinteticen mediante una combinación de multiplexores y otras compuertas lógicas.

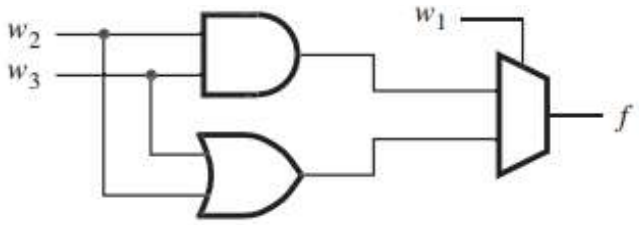
Supóngase que queremos implementar de esta forma la función original de tres entradas $f = \overline{w1}w2w3 + w1\overline{w2}w3 + w1w2\overline{w3} + w1w2w3$ $f = \overline{w1}w2w3 + w1\overline{w2}w3 + w1w2\overline{w3} + w1w2w3$. La tabla de verdad puede modificarse de tal forma que si *w1=0*, entonces *f=w2w3*, y si *w1=1*, entonces *f=w2+w3*.

w_1	w_2	w_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

w_1	f
0	w_2w_3
1	w_2+w_3

Tabla de verdad de la función de ejemplo

La función expuesta puede sintetizarse (aplicando álgebra booleana) y quedar como $f = \overline{w1}(w2w3) + w1(w2 + w3) f = \overline{w1}(w2w3) + w1(w2 + w3)$. El circuito resultante es el siguiente:



Circuito de la función implementado con un multiplexor dos a uno

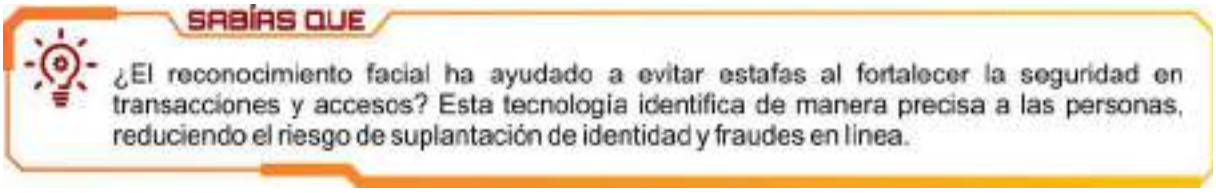
Las implementaciones de funciones lógicas con multiplexores requieren que una función se descomponga en términos de las variables empleadas como entradas de selección. Esto se logra mediante un teorema propuesto por Claude Shannon. El **Teorema de expansión de Shannon** dice que cualquier función booleana $f(w1,\ldots,wn) f(w1,\ldots,wn)$ puede escribirse en la forma $f(w1,w2,\ldots,wn) = w1 \cdot f(0,w2,\ldots,wn) + w1 \cdot f(1,w2,\ldots,wn)$ $f(w1,w2,\ldots,wn) = w1 \cdot f(0,w2,\ldots,wn) + w1 \cdot f(1,w2,\ldots,wn)$. (Brown & Vranesic, 2006).

SISTEMA DE CONTROL DE ACCESO CON RECONOCIMIENTO FACIAL

Un Sistema de Control de Acceso con Reconocimiento Facial es una solución tecnológica que permite gestionar y controlar el acceso a determinados espacios de forma segura mediante la identificación biométrica. Este sistema utiliza cámaras y algoritmos avanzados de reconocimiento facial para verificar la identidad de las personas, comparando sus características faciales con una base de datos preestablecida.

Los sistemas de reconocimiento facial quedan incluidos dentro de la categoría de reconocimiento biométrico. Estas metodologías de identificación toman un papel cada vez más significativo en ámbitos relacionados con la seguridad, administración e industrial. En los métodos de biometría, el reconocimiento facial ha atraído un creciente interés debido a que permite identificar y verificar la identidad de individuos de una manera discreta y no intrusiva. (Fontalba Roca, 2024).

El funcionamiento se basa en la captura de la imagen del rostro de una persona al acercarse a un punto de acceso. El sistema analiza rasgos únicos, como la distancia entre los ojos, la forma de la nariz o el contorno del rostro, para generar un patrón facial que se compara con los registros existentes. Si la persona es identificada y está autorizada, se le permite el acceso; de lo contrario, se deniega la entrada.



Este tipo de tecnología es cada vez más utilizado en edificios inteligentes, empresas, aeropuertos y otros lugares donde la seguridad es primordial. Además, al eliminar la necesidad de tarjetas, contraseñas o huellas dactilares, el reconocimiento facial ofrece una solución más conveniente, rápida y difícil de falsificar.

En la actualidad es un tema importante debido a los avances tecnológicos y su impacto en la seguridad personal, la privacidad y la confianza en los sistemas de autenticación. (Cardenas, 2023.)

En el contexto del aprendizaje de la electrónica, antes de llevar un producto al mercado es fundamental realizar un proceso de prototipado. En la enseñanza de esta disciplina, los microcontroladores como Arduino desempeñan un papel crucial. No obstante, antes de ensamblar un prototipo físico, es imprescindible pasar por una fase de simulación. Para ello, contamos con una variedad de plataformas y software que ofrecen una extensa gama de componentes electrónicos simulables, junto con sus respectivos compiladores de código. Este enfoque garantiza que los diseños sean probados y optimizados antes de su implementación real, minimizando errores y mejorando la eficiencia del desarrollo.

Es importante señalar que los simuladores no siempre permiten emular todas las acciones o componentes reales. En estos casos, la creatividad es clave. Por ejemplo, si no se puede simular una puerta, podemos representarla con un servomotor, adaptando el diseño de manera eficiente y práctica.

DISEÑO DEL PROTOTIPO

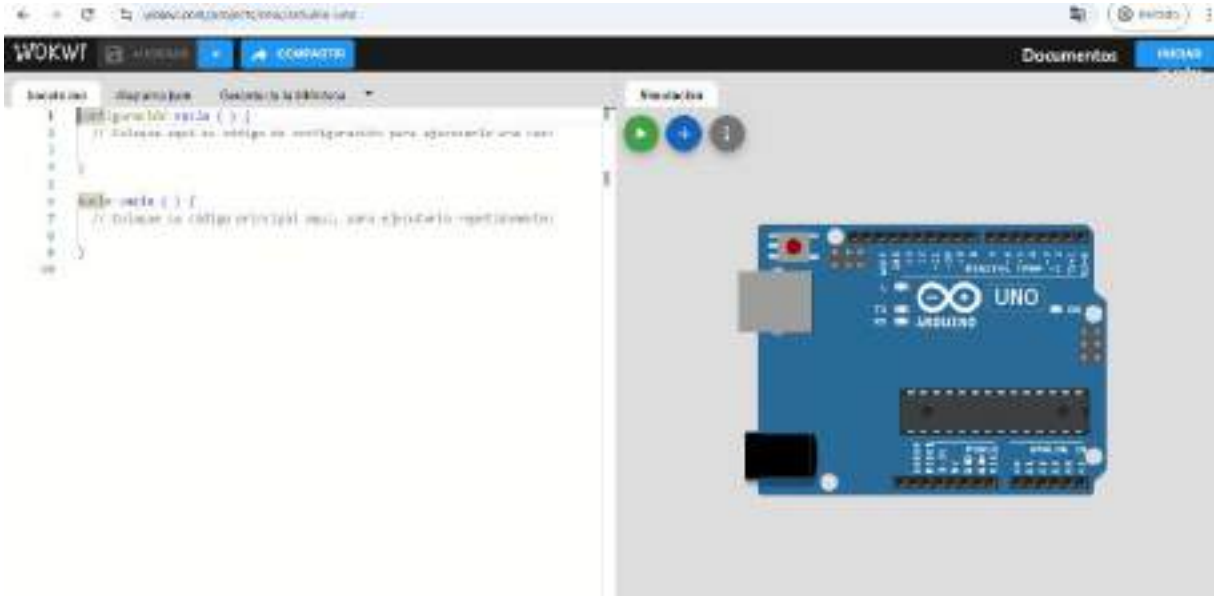
Para el **diseño** de esta **práctica**, vamos a usar una plataforma gratuita que podemos encontrar en la web, como **Wokwi.com**. Así como también podemos usar **Tinkercad**, en estas plataformas podemos encontrar una variedad de componentes, lo que facilita la elaboración de nuestra práctica.

Hagamos un breve recorrido por Wokwi:

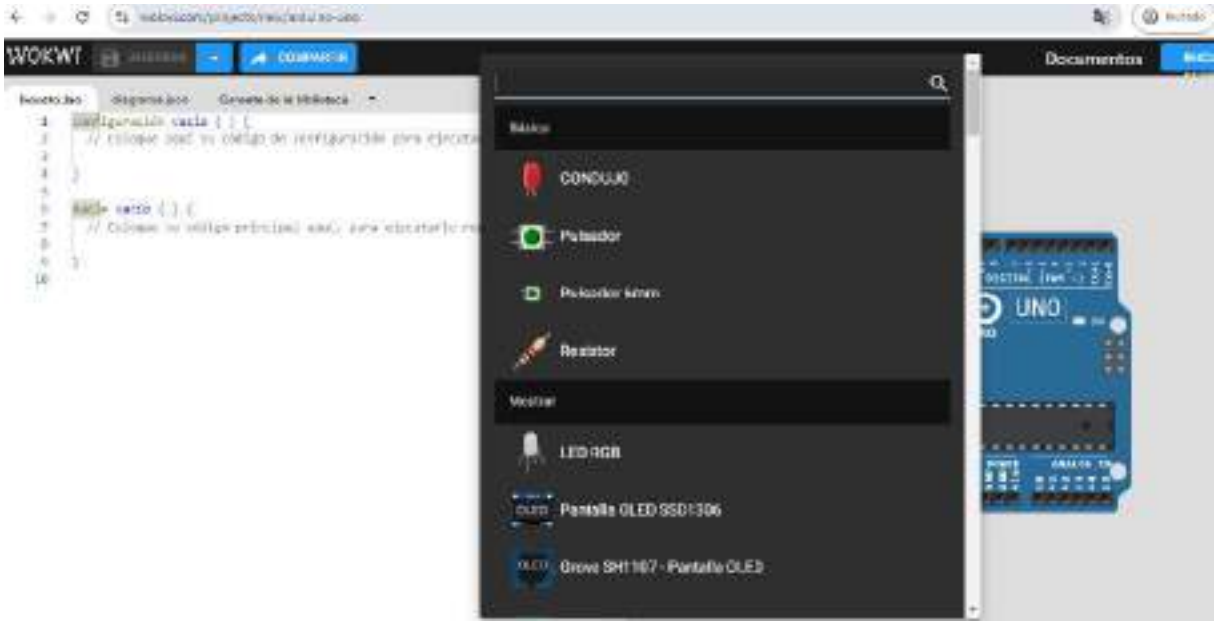
- Abrimos en nuestro navegador Wokwi.com y visualizamos la amplia variedad de microcontroladores.

- Damos clic en **Arduino** y se nos abre otra pestaña. Luego, nos dirigimos a la sección “Empezar desde cero”.

- Una vez allí, podemos escoger nuestro microcontrolador.



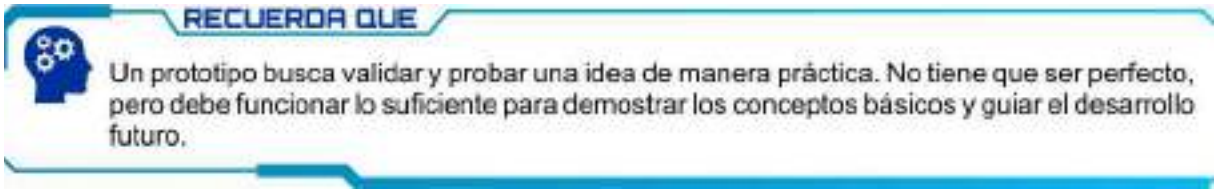
- Ya podemos visualizar la sección de compilación de nuestro código, y del otro lado tenemos la simulación de nuestro Arduino. Justo en esa sección, al hacer clic en el ícono, podremos agregar más componentes a nuestro proyecto.



Ahora que ya sabemos cómo usar esta plataforma, comenzaremos con nuestra práctica.

Descripción del Prototipo:

El prototipo implementado con el ESP32 simula un sistema de control de acceso donde un servomotor representa una puerta, y un botón simula la activación de una cámara de seguridad. Al presionar el botón (simulando la cámara), la "puerta" se abre durante un periodo determinado, y luego se cierra automáticamente.



Componentes Utilizados:

Componentes	Cantidad	Especificaciones
ESP32	1	Microcontrolador principal que gestiona el sistema.
Servomotor	1	Simula la apertura y cierre de una puerta.
Botón	1	Representa la activación de la cámara de seguridad que dispara el mecanismo de la puerta.

Simulación del Comportamiento:

- **Servomotor (puerta)**: El servomotor se controla mediante señales PWM para simular la apertura (de 0° a 180°) y cierre (de 180° a 0°) de una puerta.
- **Botón (cámara)**: Al presionar el botón, que simula la cámara de seguridad, se activa el proceso de apertura de la puerta. Este enfoque permite probar el sistema sin necesidad de una cámara real.

Pasos para la elaboración de la práctica:

vamos a preparar nuestros componentes para nustreo proyecto para eso usaremos la plataforma wokwi.

Una vez que estemos en la plataforma Wokwi.com, vamos a escoger los siguientes componentes:

- ESP32

Figura 1: microcontrolador ESP32

SABÍAS QUE

El ESP32 es un microcontrolador muy popular en el ámbito de IoT (Internet de las Cosas) por su capacidad de conectividad Wi-Fi y Bluetooth integradas? Esto lo convierte en una opción ideal para proyectos inalámbricos y dispositivos inteligentes.

- Servomotor

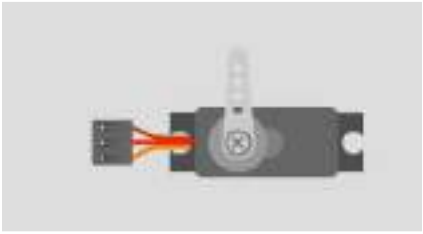


Figura 2: servomotor usado para simular una puerta

- Botón

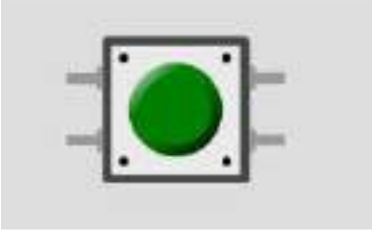


Figura 3: botón o pulsador usado para simular una cámara

Una vez que tengamos los componentes, estaremos listos para comenzar a trabajar.

Figura 4: interfaz lista con componentes necesarios

Desarrollo del código:

Ahora vamos a desarrollar el código de nuestro proyecto, el cual dará las instrucciones que debe seguir nuestro prototipo.

Asegúrate de seguir cada paso con atención y en el orden indicado. Una conexión incorrecta o un código mal escrito puede hacer que el proyecto no funcione como se espera. Tómate tu tiempo para revisar las conexiones y el código antes de realizar la simulación.

Código

Estas líneas del código definen los pines digitales de la placa a los que se conectan el servo motor y el botón. servoPin está asignado al pin 13 y botonPin al pin 12.

```
const int servoPin = 13; // Pin del servo
const int botonPin = 12; // Pin del botón
```

Estas líneas configuran los parámetros del pulso PWM para el servo motor:

- **servoMinPulse**: Ancho del pulso mínimo de 1000 microsegundos (1 ms), que mueve el servo al extremo inferior de su rango.
- **servoMaxPulse**: Ancho del pulso máximo de 2000 microsegundos (2 ms), que mueve el servo al extremo superior de su rango.
- **servoPeriod**: Periodo total del PWM de 20000 microsegundos (20 ms), que define la frecuencia de la señal PWM.

```
// Configuración de PWM
const int servoMinPulse = 1000; // Ancho del pulso mínimo en
microsegundos (1 ms)
const int servoMaxPulse = 2000; // Ancho del pulso máximo en
microsegundos (2 ms)
const int servoPeriod = 20000; // Período de PWM en microsegundos (20 ms)
```

void setup(): Configura el entorno inicial.

pinMode(servoPin, OUTPUT); define el pin del servo como salida.

pinMode(botonPin, INPUT_PULLUP); configura el pin del botón como entrada con resistencia pull-up, lo que significa que el pin está normalmente en alto (HIGH) y baja (LOW) cuando el botón se presiona.

void loop(): Se ejecuta continuamente.

Lee el estado del botón con digitalRead(botonPin).

Si el botón está presionado (botonEstado == LOW), imprime “Acceso Autorizado”, llama a abrirPuerta(), mantiene la puerta abierta por 5 segundos con delay(5000), y luego llama a cerrarPuerta().

Si el botón no está presionado, imprime “Esperando acceso...”.

delay(100); introduce una pequeña espera para evitar lecturas erráticas del botón.

```
void setup() {
  pinMode(servoPin, OUTPUT);
  pinMode(botonPin, INPUT_PULLUP); // Configura el botón con resistencia
pull-up
}

void loop() {
  // Lee el estado del botón
  int botonEstado = digitalRead(botonPin);

  if (botonEstado == LOW) {  // Si el botón está presionado
    Serial.println("Acceso Autorizado");
    abrirPuerta();
    delay(5000); // Puerta abierta por 5 segundos
    cerrarPuerta();
  } else {
    Serial.println("Esperando acceso...");
  }

  delay(100); // Espera corta para evitar lecturas erráticas
}
```

void abrirPuerta(): Esta función mueve el servo motor para abrir la puerta.

for (int i = 0; i < 180; i++): Un bucle que aumenta el valor de i de 0 a 179, lo que representa el ángulo de movimiento del servo desde 0 hasta 180 grados.

int pulseWidth = map(i, 0, 180, servoMinPulse, servoMaxPulse);: Mapea el valor de i (que varía entre 0 y 180) a un ancho de pulso entre servoMinPulse (1 ms) y servoMaxPulse (2 ms).

digitalWrite(servoPin, HIGH);: Establece el pin del servo en alto para iniciar el pulso.

delayMicroseconds(pulseWidth);: Mantiene el pin en alto durante el tiempo calculado por pulseWidth, ajustando la posición del servo.

digitalWrite(servoPin, LOW);: Establece el pin del servo en bajo para terminar el pulso.

delayMicroseconds(servoPeriod - pulseWidth);: Mantiene el pin en bajo durante el resto del periodo PWM para completar el ciclo.

Esta función mueve el servo motor gradualmente desde la posición de 0 hasta 180 grados, lo que simula la apertura de una puerta.

```
// Función para abrir la puerta
void abrirPuerta() {
  for (int i = 0; i < 180; i++) {
    int pulseWidth = map(i, 0, 180, servoMinPulse, servoMaxPulse);
    digitalWrite(servoPin, HIGH);
    delayMicroseconds(pulseWidth);
    digitalWrite(servoPin, LOW);
    delayMicroseconds(servoPeriod - pulseWidth);
  }
}
```

void cerrarPuerta(): Esta función mueve el servo motor para cerrar la puerta.

for (int i = 180; i >= 0; i--): Un bucle que disminuye el valor de i de 180 a 0, lo que representa el ángulo de movimiento del servo desde 180 hasta 0 grados.

int pulseWidth = map(i, 0, 180, servoMinPulse, servoMaxPulse);: Mapea el valor de i (que varía entre 180 y 0) a un ancho de pulso entre servoMinPulse (1 ms) y servoMaxPulse (2 ms).

digitalWrite(servoPin, HIGH);: Establece el pin del servo en alto para iniciar el pulso.

delayMicroseconds(pulseWidth);: Mantiene el pin en alto durante el tiempo calculado por pulseWidth, ajustando la posición del servo.

digitalWrite(servoPin, LOW);: Establece el pin del servo en bajo para terminar el pulso.

delayMicroseconds(servoPeriod - pulseWidth);: Mantiene el pin en bajo durante el resto del periodo PWM para completar el ciclo.

Esta función mueve el servo motor gradualmente desde la posición de 180 grados hasta 0 grados, simulando el cierre de una puerta.

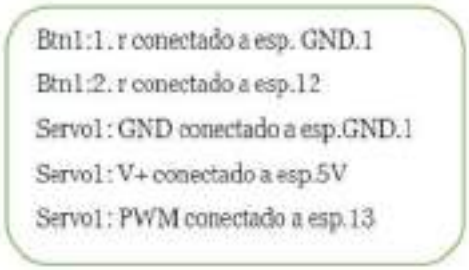
```
// Función para cerrar la puerta
void cerrarPuerta() {
  for (int i = 180; i >= 0; i--) {
    int pulseWidth = map(i, 0, 180, servoMinPulse, servoMaxPulse);
    digitalWrite(servoPin, HIGH);
    delayMicroseconds(pulseWidth);
    digitalWrite(servoPin, LOW);
    delayMicroseconds(servoPeriod - pulseWidth);
  }
}
```

Conexiones de los componentes electrónicos:

Una vez que tenemos listo nuestro código, es importante conectar correctamente nuestros componentes, ya que, si lo hacemos mal, el código no funcionará como debería.

Primero, debes saber que para conectar los componentes solo tienes que hacer clic en un pin y luego en el pin que te indican los pasos. ¡Fácil, verdad!

Al acercar el puntero de tu mouse podrás visualizar el nombre de cada pin y así te será más fácil identificar los pines.



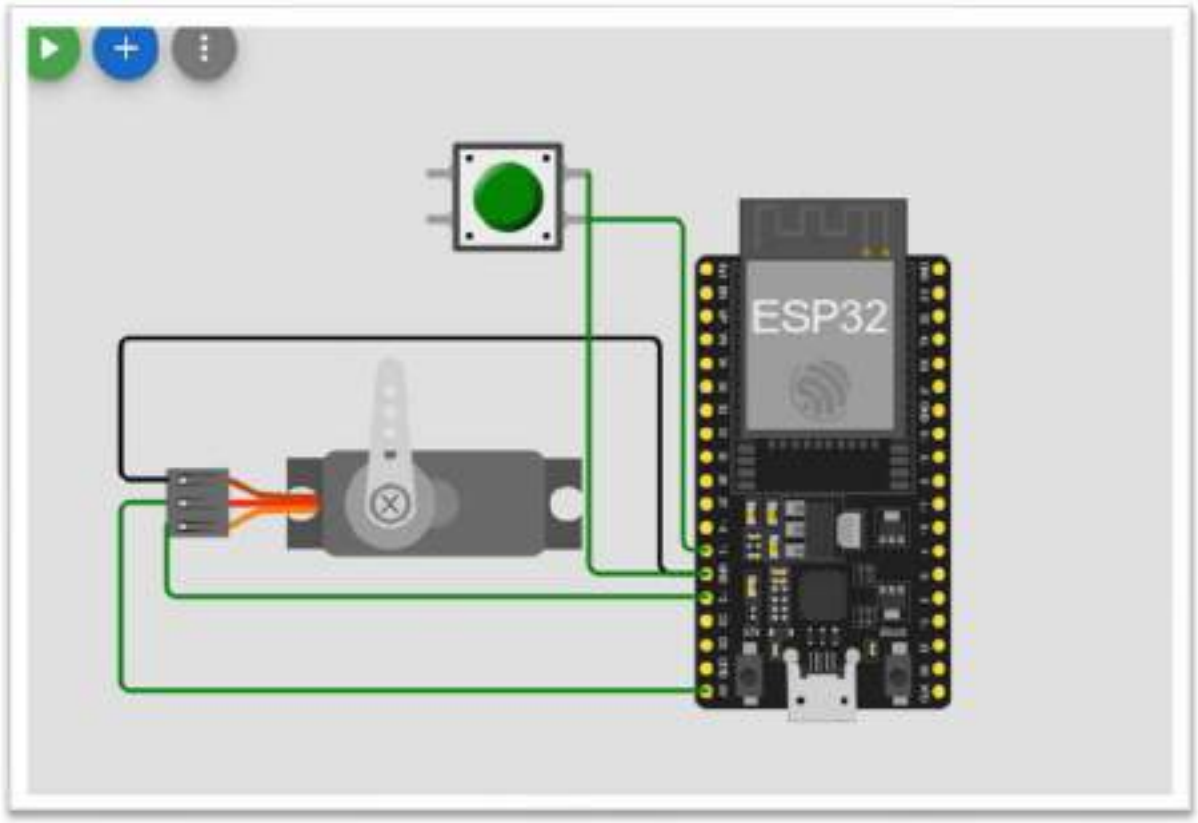


Figura 5: Componentes conectados correctamente y en orden

Prueba final:

Has llegado a la parte final de esta práctica. Ahora solo debes compilar y ejecutar tu proyecto, y verificar que todo funcione como debe.

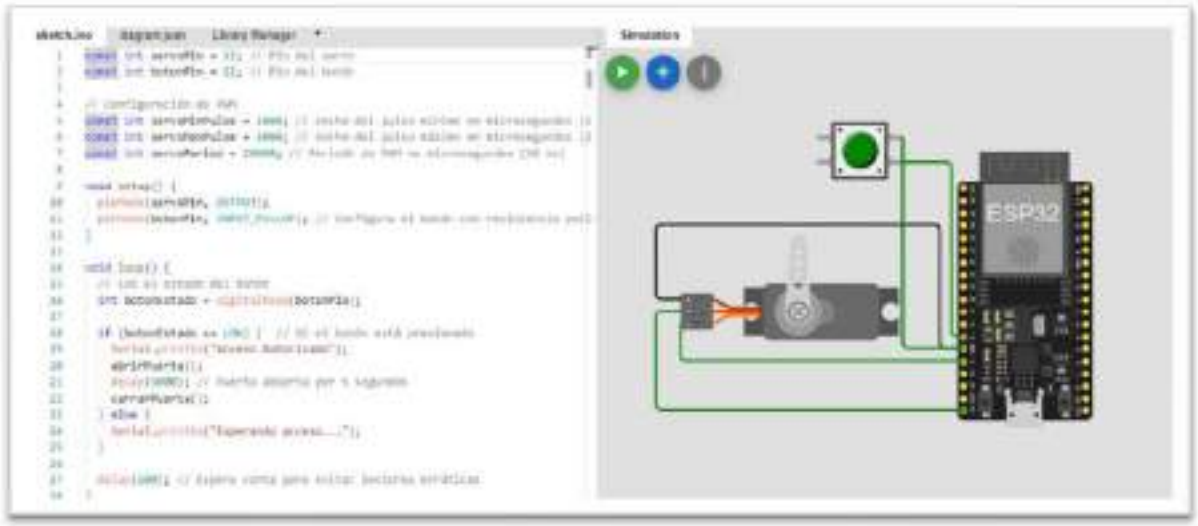


Figura 6: resultado final del circuito con su código programado

Ejecutar nuestro proyecto

Para eso, solo debemos hacer clic en el ícono,  y nuestro código se ejecutará sin problemas. Ahora, solo debes presionar el botón para simular la acción de una cámara, y podrás ver cómo el servo motor se mueve, simulando la apertura de la puerta.

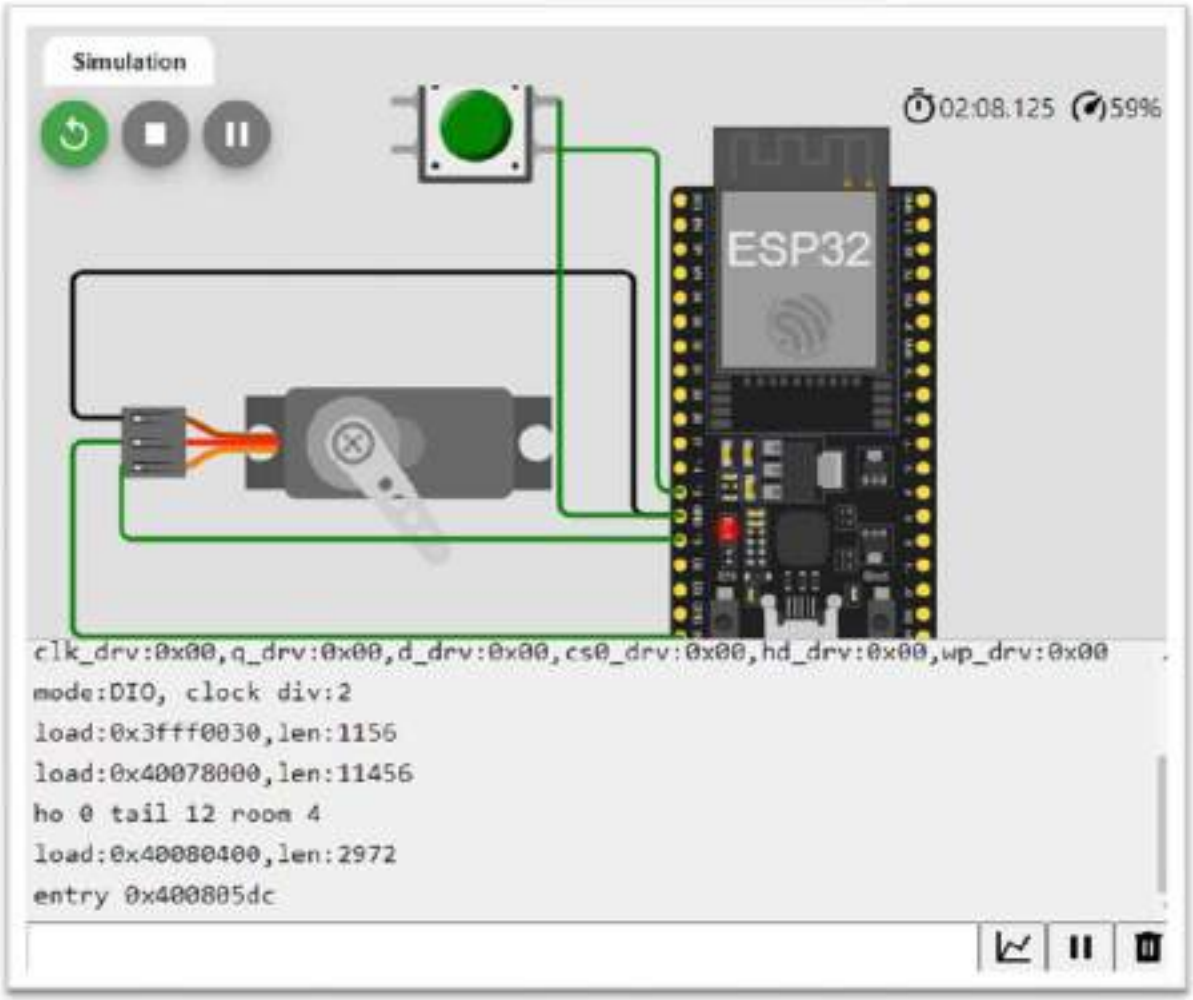


Figura 7: programa funcionando correctamente con los componentes

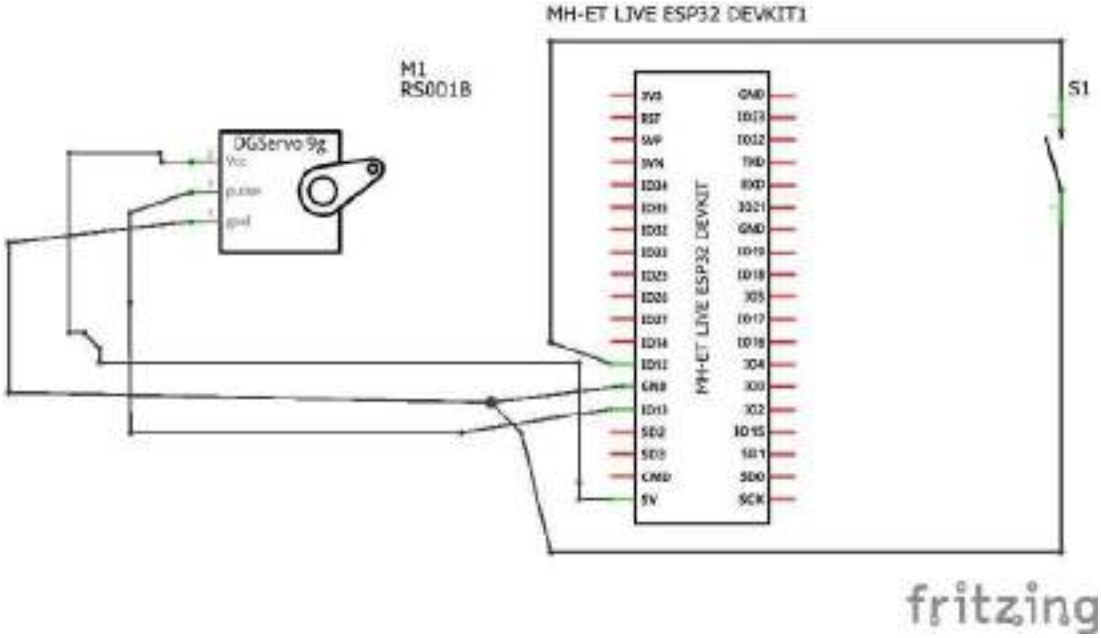


Figura 8: Sistema de Control de Acceso con Reconocimiento Facial
usando una cámara ESP32-CAM y un servo motor para simular una puerta

PRÁCTICAS DE SISTEMAS ELECTRÓNICOS

SISTEMA DE MONITOREO DE CALIDAD DEL AIRE: CON SENSORES DE CO2 Y PM2.5, Y CONECTIVIDAD A LA NUBE

El Sistema de Monitoreo de Calidad del Aire se ha vuelto esencial en entornos urbanos e industriales, donde la contaminación puede afectar la salud y el bienestar de las personas. La integración de sensores de CO2 y PM2.5 permite medir de forma precisa los niveles de dióxido de carbono y partículas finas en el aire, dos indicadores clave de la calidad ambiental. Este sistema proporciona datos en tiempo real para evaluar las condiciones del aire y tomar medidas inmediatas en caso de que se superen los límites seguros.

El uso de sensores de CO2 ayuda a monitorear la concentración de dióxido de carbono, lo que es crucial en espacios cerrados como oficinas y hogares. Altos niveles de CO2 pueden indicar una ventilación insuficiente y afectar la productividad y salud de las personas. Por otro lado, los sensores de PM2.5 detectan partículas finas en el aire que pueden penetrar en los pulmones y causar problemas respiratorios. La combinación de estos sensores ofrece una visión integral de la calidad del aire en un espacio determinado.

La conectividad a la nube es un componente clave de este sistema, ya que permite almacenar y analizar los datos recopilados de forma remota. Esto facilita el seguimiento continuo y la generación de alertas si los niveles de contaminación alcanzan valores peligrosos. Además, la integración con la nube permite a los usuarios acceder a los datos desde cualquier lugar y en cualquier momento, ayudando a tomar decisiones informadas para mejorar la calidad del aire y, en última instancia, la calidad de vida.

SABÍAS QUE

¿Un Sistema de Monitoreo de Calidad del Aire usa sensores para medir niveles de CO2 y partículas PM2.5 en tiempo real? Esto permite identificar y reaccionar a problemas de contaminación de manera rápida y eficaz.

DISEÑO DEL PROTOTIPO

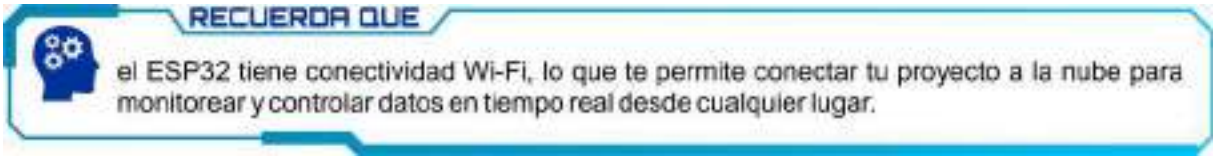
Para el diseño de este prototipo usaremos la plataforma **Wokwi.com**

Descripción del Prototipo:

El fin de este prototipo es monitorear la calidad del aire mediante sensores de CO2 y PM2.5, los cuales serán simulados con potenciómetros.

Componentes Utilizados:

Componentes	Cantidad	Descripción
ESP32	1	Microcontrolador principal que gestiona el sistema.
Potenciómetro	2	Un potenciómetro es un resistor variable con tres terminales, que permite ajustar manualmente la resistencia y controlar el voltaje o la corriente en un circuito.



Simulación del Comportamiento:

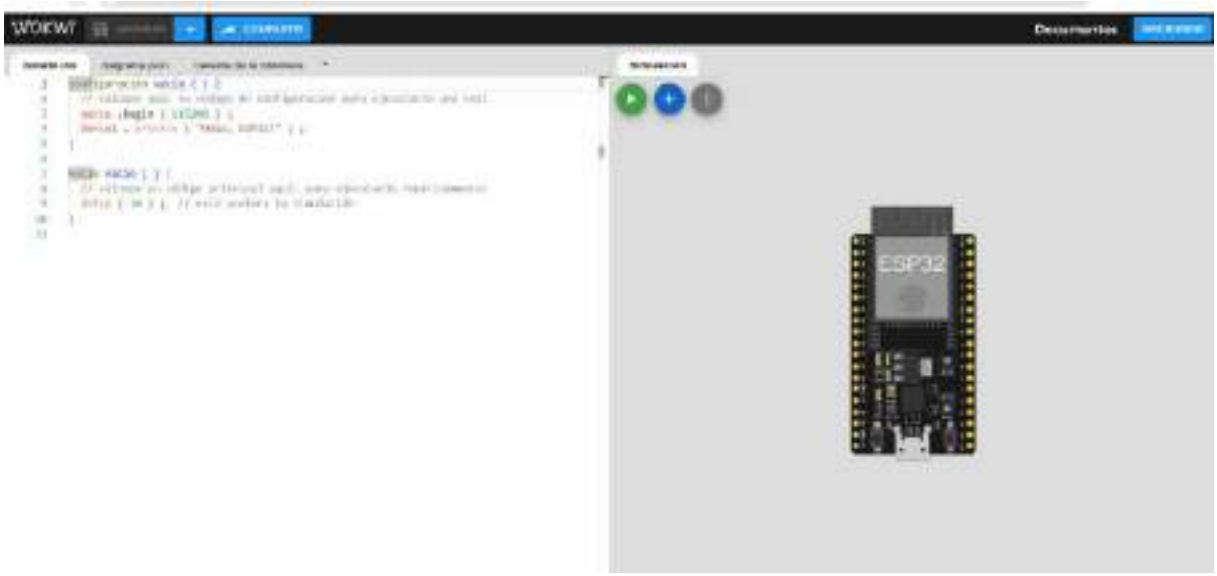
- **Potenciómetro (CO2):** El potenciómetro conectado al pin analógico 34 simula los niveles de CO2 en el aire. Al girarlo, se cambia el valor analógico, que se mapea a una concentración de CO2 en un rango de 0 a 5000 ppm (partes por millón). Esto permite observar cómo varía la calidad del aire según la concentración de CO2.
- **Potenciómetro (PM2.5):** El potenciómetro conectado al pin analógico 35 simula los niveles de partículas PM2.5 en el aire. Al ajustar este potenciómetro, se obtienen valores que se traducen a concentraciones en un rango de 0 a 500 µg/m³ (microgramos por metro cúbico). Esta simulación facilita el monitoreo de la calidad del aire sin necesidad de sensores especializados.

Pasos para la elaboración de la práctica:

Vamos a preparar nuestros componentes para nuestro proyecto para eso usaremos la plataforma wokwi.

Una vez que estemos en la plataforma Wokwi.com, vamos a escoger los siguientes componentes:

- ESP32



- Potenciómetro

Una vez que tengamos los componentes, estaremos listos para comenzar a trabajar.

Desarrollo del código:

Ahora vamos a desarrollar el código de nuestro proyecto, el cual dará las instrucciones que debe seguir nuestro prototipo.

Asegúrate de seguir cada paso con atención y en el orden indicado. Una conexión incorrecta o un código mal escrito puede hacer que el proyecto no funcione como se espera. Tómate tu tiempo para revisar las conexiones y el código antes de realizar la simulación.

Código

Estas líneas del código definen los pines analógicos a los que se conectan los potenciómetros para simular los sensores de calidad del aire:

- **const int co2Pin = 34;**: Define el pin analógico 34 como la entrada para el potenciómetro que simula los niveles de CO2.
- **const int pm25Pin = 35;**: Define el pin analógico 35 como la entrada para el potenciómetro que simula los niveles de partículas PM2.5.

Estos potenciómetros permiten ajustar manualmente los niveles de CO2 y PM2.5, representando cambios en la calidad del aire de forma controlada.

```
const int co2Pin = 34; // Pin analógico para el potenciómetro de CO2
const int pm25Pin = 35; // Pin analógico para el potenciómetro de PM2.5
```

La **función setup**() se ejecuta una vez cuando el programa comienza a ejecutarse. Dentro de esta función, se encuentra la siguiente línea:

- **Serial.begin(115200);**: Inicializa la comunicación serial a una velocidad de 115200 baudios. Esto permite enviar y recibir datos entre la placa y un ordenador o dispositivo conectado. En este caso, se utilizará para imprimir y monitorear los valores de los potenciómetros, que simulan los niveles de CO2 y PM2.5, a través del monitor serial.

```
void setup() {
  Serial.begin(115200); // Inicializa la comunicación serial
```

Esta es la **función loop**(), que se ejecuta continuamente en el programa. A continuación, te explico el funcionamiento de cada parte:

1. **Lectura de los potenciómetros:**

   - **int co2Value = analogRead(co2Pin);**: Lee el valor analógico del potenciómetro conectado al pin co2Pin (pin 34), simulando el nivel de CO2.
   - **int pm25Value = analogRead(pm25Pin);**: Lee el valor analógico del potenciómetro conectado al pin pm25Pin (pin 35), simulando el nivel de partículas PM2.5.

2. **Mostrar los valores en la consola serial:**

   - **Serial.print("CO2: ");**: Imprime el texto "CO2: " en el monitor serial.
   - **Serial.print(co2Value);**: Imprime el valor leído del potenciómetro de CO2.
   - **Serial.print(" PM2.5: ");**: Imprime el texto " PM2.5: " en el monitor serial.
   - **Serial.println(pm25Value);**: Imprime el valor leído del potenciómetro de PM2.5 y hace un salto de línea para la siguiente lectura.

3. **Esperar 1 segundo:**

   - **delay(1000);**: Pausa la ejecución del programa durante 1000 milisegundos (1 segundo) antes de repetir el ciclo, lo que permite ver claramente las lecturas en la consola serial.

Este código lee los valores de los potenciómetros que simulan los niveles de CO2 y PM2.5 y los muestra en la consola serial cada segundo.

```
void loop() {
  int co2Value = analogRead(co2Pin); // Lee el valor del
potenciómetro de CO2
  int pm25Value = analogRead(pm25Pin); // Lee el valor del
potenciómetro de PM2.5

  // Mostrar los valores en la consola serial
  Serial.print("CO2: ");
  Serial.print(co2Value);
  Serial.print(" PM2.5: ");
  Serial.println(pm25Value);

  delay(1000); // Espera de 1 segundo
}
```

Conexiones de los componentes electrónicos:

Una vez que tenemos listo nuestro código, es importante conectar correctamente nuestros componentes, ya que, si lo hacemos mal, el código no funcionará como debería.

Es muy fácil tenemos que conectar de la misma forma que lo hicimos en la primera práctica.

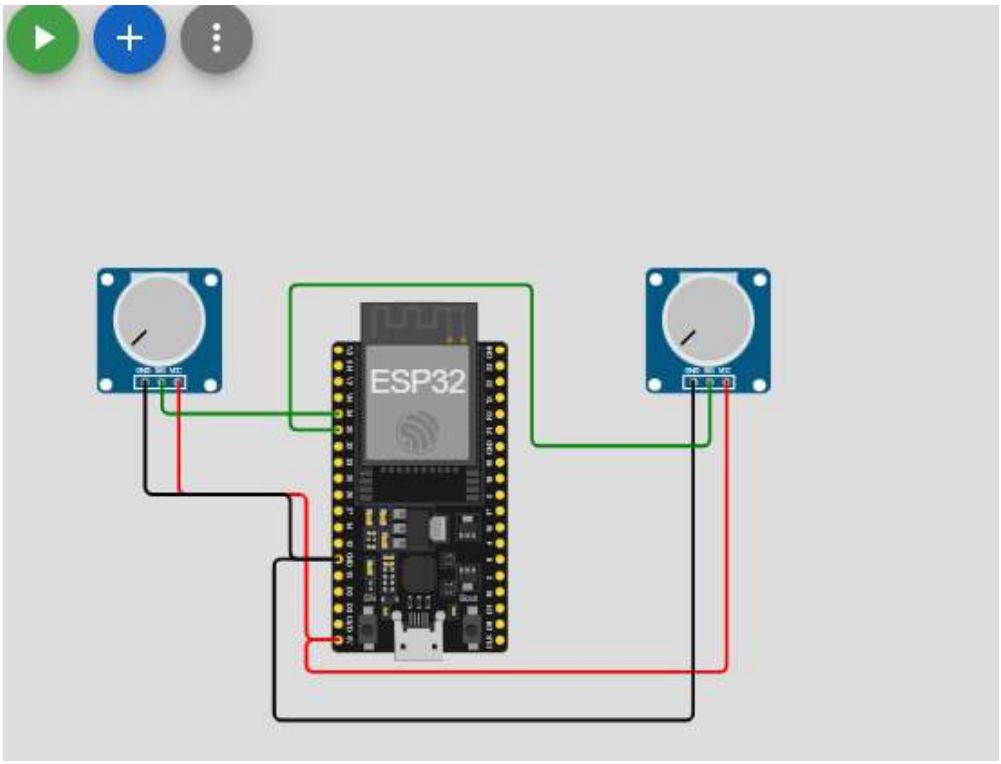


Figura 9: prototipo correctamente conectado como se recomienda en los pasos

Prueba final:

Has llegado a la parte final de esta práctica. Ahora solo debes compilar y ejecutar tu proyecto, y verificar que todo funcione como debe.

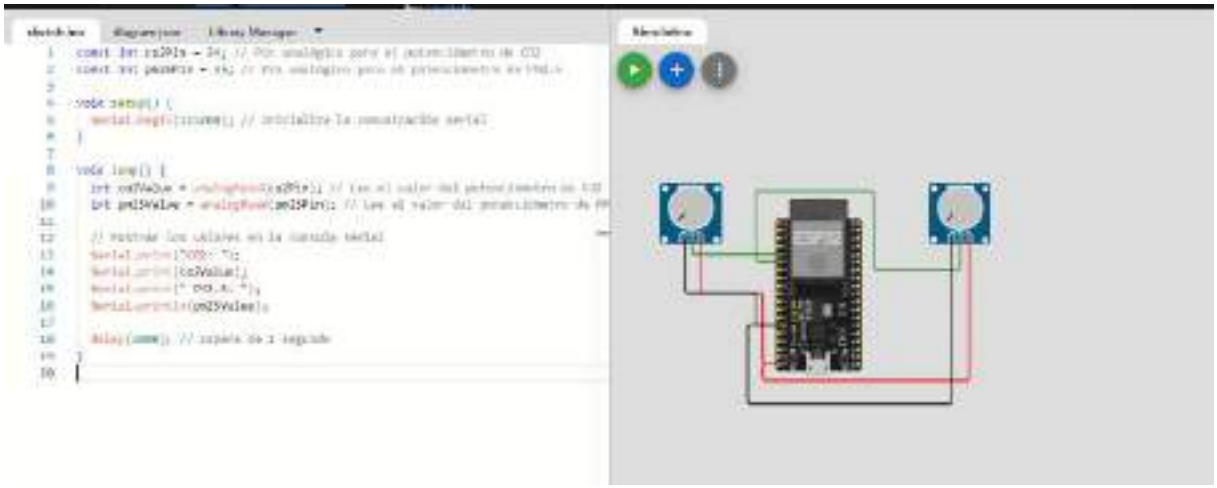


Figura 10: código y prototipo listo para simular

Ejecutar nuestro proyecto

Cuando hagas clic en el ícono de ejecutar, tus potenciómetros, que simulan los sensores, mostrarán una lectura de cero. Debes ajustar los potenciómetros para que los valores varíen, simulando así las lecturas cambiantes de los sensores.

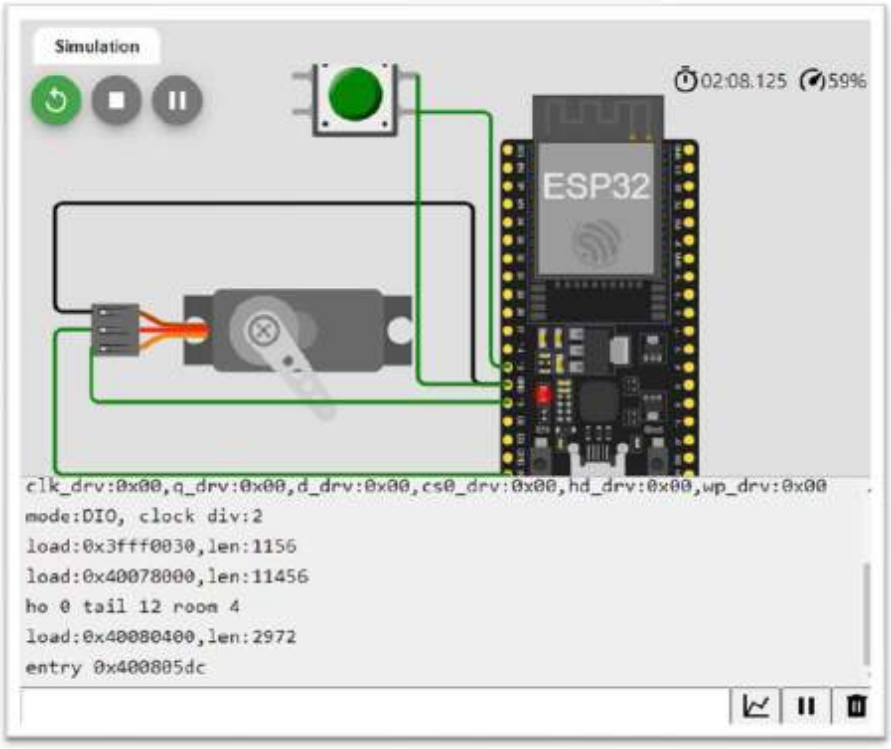


Figura 11: ejecución de código correctamente y sensores simulados marcando en cero

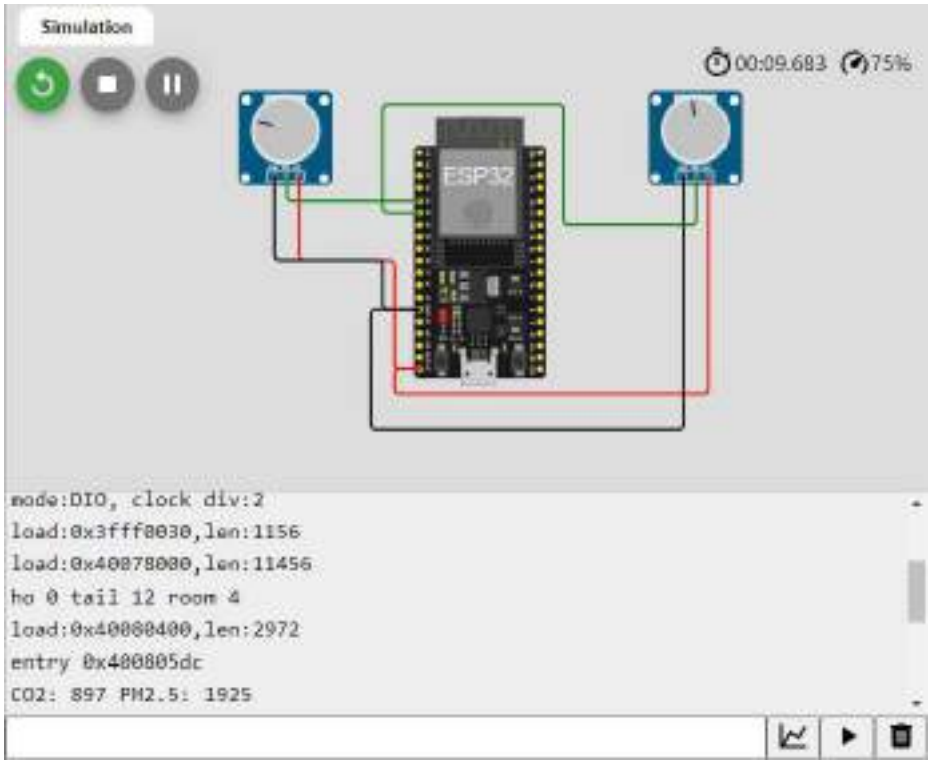


Figura 12: ejecución de código correctamente y sensores simulados se mueven para poder medir la calidad del aire y sus valores cambian

DATOS ÚTILES

Los niveles de CO2 en el aire son un buen indicador de la calidad del aire interior. Concentraciones superiores a 1000 ppm pueden indicar una ventilación inadecuada y posibles riesgos para la salud, como dolores de cabeza y fatiga.

PRÁCTICAS DE SISTEMAS ELECTRÓNICOS

CONTROL DE PUERTA DE GARAJE AUTOMÁTICO: USANDO SENSORES DE PROXIMIDAD PARA ABRIR Y CERRAR LA PUERTA

En la era de la automatización, los sistemas de control de puertas de garaje automáticas han ganado popularidad por su comodidad y seguridad. Estos sistemas utilizan sensores de proximidad para detectar la presencia de vehículos o personas cerca de la puerta, permitiendo su apertura o cierre sin necesidad de intervención manual. Esta tecnología no solo simplifica el acceso al garaje, sino que también mejora la seguridad al evitar la necesidad de salir del vehículo para operar la puerta.

Los sensores de proximidad son cruciales en estos sistemas, ya que detectan la presencia y la distancia de objetos o personas mediante ondas ultrasónicas o infrarrojas. Cuando un vehículo se aproxima, los sensores envían señales al controlador del sistema para activar el motor que abre la puerta. De igual manera, el sistema puede detectar la ausencia del vehículo y cerrar la puerta automáticamente, asegurando que el garaje quede seguro cuando no se está utilizando.

SABÍAS QUE

¿Un garaje automático puede facilitar la vida de los adultos mayores al proporcionarles mayor comodidad y seguridad? Gracias a sensores y sistemas automatizados, pueden evitar el esfuerzo físico de abrir y cerrar puertas manualmente, reduciendo el riesgo de accidentes y haciéndoles sentir más independientes en su propio hogar.

La implementación de un control automático de puerta de garaje no solo proporciona conveniencia, sino que también contribuye a la eficiencia energética y la seguridad del hogar. Los sensores garantizan que la puerta se abra solo cuando es necesario, evitando el desgaste innecesario del motor y proporcionando una solución moderna y efectiva para el control de accesos en el hogar.

DISEÑO DEL PROTOTIPO

Para el diseño de este prototipo usaremos la plataforma **Wokwi.com** en esta simulación vamos a subir un poco el nivel de complejidad usando más componentes electrónicos y más especificaciones técnicas.

Descripción del Prototipo:

El prototipo de Control de Puerta de Garaje Automático utiliza sensores de proximidad para gestionar la apertura y cierre de la puerta del garaje. Los sensores detectan la presencia de vehículos o personas cerca de la puerta, activando

el mecanismo de apertura o cierre automáticamente. Este sistema mejora la comodidad y la seguridad al permitir un acceso sin contacto físico y al asegurar que la puerta se maneje de manera eficiente.

Componentes Utilizados:

Componentes	Cantidad	Descripción
Arduino Uno	1	Microcontrolador principal que gestiona el sistema.
HC-SR04 Ultrasonic Distance Sensor	1	Es un sensor que mide distancias utilizando ondas ultrasónicas. Emite pulsos de sonido y calcula la distancia al medir el tiempo que tardan en regresar.
LED	1	Es un componente electrónico que emite luz cuando pasa una corriente eléctrica a través de él.
Resistencia 1K ohm	1	Es un componente eléctrico que limita la cantidad de corriente que pasa a través de un circuito.
Micro Servo Motor	1	Simula la apertura y cierre de una puerta.

Simulación del Comportamiento:

HC-SR04 Ultrasonic Distance Sensor:

- El sensor se dispara mediante el pin trigPin y se lee la duración del pulso en el pin echoPin. Esta duración se convierte en una distancia que se usa para decidir cuándo abrir o cerrar la puerta.

LED:

- Indica visualmente el estado de la puerta (abierta o cerrada). El LED se enciende cuando la puerta está abierta y se apaga cuando está cerrada.
- Se controla mediante el pin ledPin. El LED se enciende en la función abrirPuerta() y se apaga en la función cerrarPuerta() para proporcionar una señal visual del estado de la puerta.

Resistencia de 1K ohm:

- La resistencia se usa para proteger el LED y cualquier otro componente conectado a los pines de entrada/salida, asegurando un funcionamiento seguro del circuito.

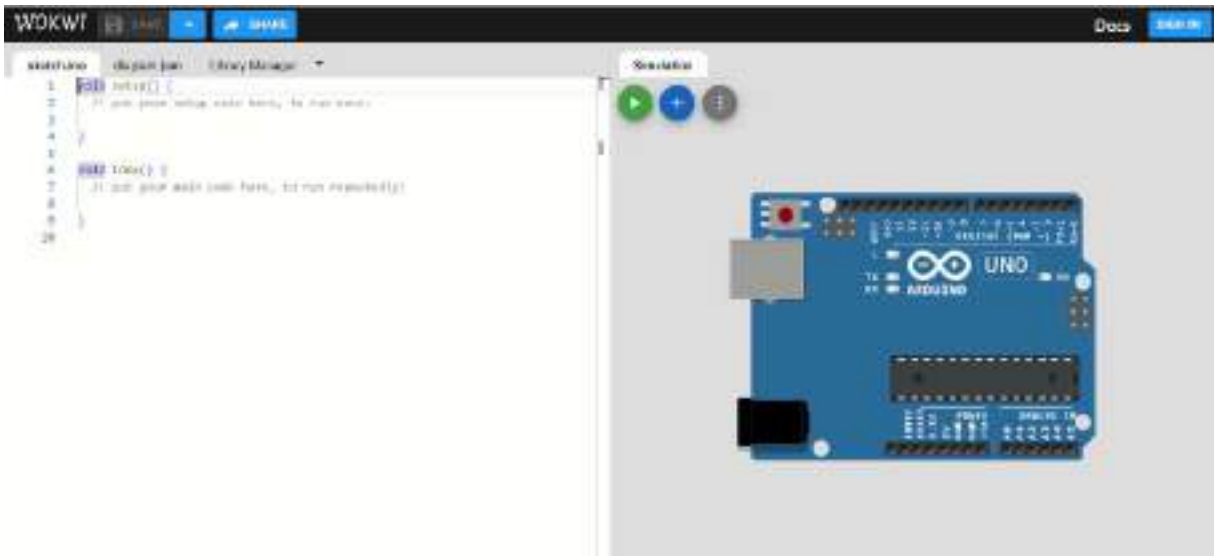
Micro Servo Motor:

- El servomotor se mueve entre las posiciones closedPosition y openPosition para simular la apertura y cierre de la puerta en función de la distancia detectada por el sensor. La velocidad de movimiento se ajusta mediante los retrasos en el bucle de control del servomotor.

Pasos para la elaboración de la práctica:

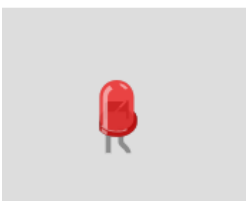
Vamos a preparar nuestros componentes para el proyecto utilizando la plataforma Wokwi.

- Arduino Uno

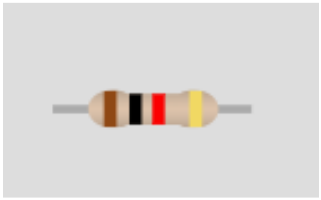


- HC-SR04 Ultrasonic Distance Sensor

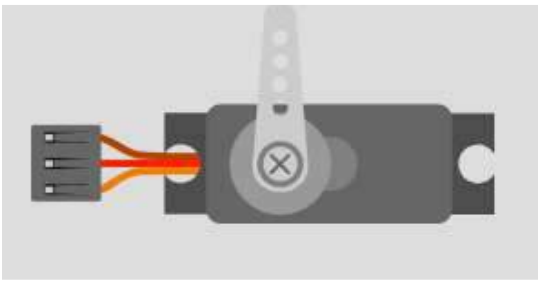
- LED



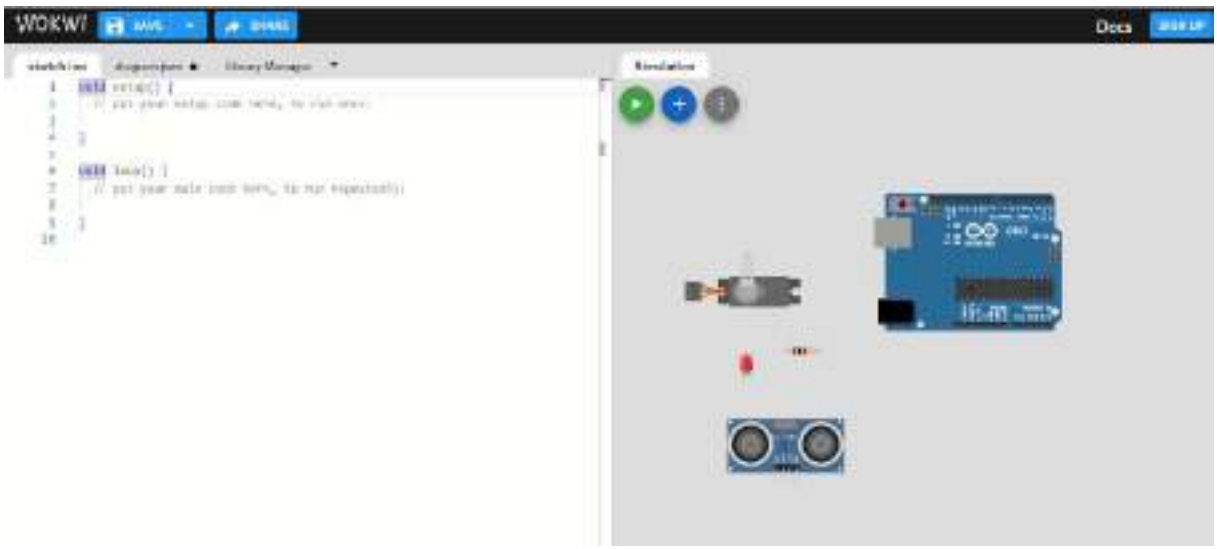
- Resistencia 1K ohm



- Micro Servo Motor



Una vez que tengamos los componentes, estaremos listos para comenzar a trabajar.



Desarrollo del código:

Ahora vamos a desarrollar el código de nuestro proyecto, que proporcionará las instrucciones necesarias para que nuestro prototipo funcione correctamente. Asegúrate de seguir cada paso con atención y en el orden indicado. Una conexión incorrecta o un código mal escrito puede impedir que el proyecto funcione como se espera. Tómate el tiempo necesario para revisar tanto las conexiones como el código antes de realizar la simulación.

Código

Define las constantes y configura los pines para los componentes que se usarán en el control automático de la puerta de garaje.

#include <Servo.h>:

Incluye la biblioteca para controlar el servomotor, permitiendo un control preciso de su posición.

Definición de pines:

- **const int trigPin = 8;**: Define el pin 8 como el pin de disparo (trig) del sensor ultrasónico HC-SR04. Este pin envía un pulso ultrasónico.
- **const int echoPin = 9;**: Define el pin 9 como el pin de eco (echo) del sensor ultrasónico. Este pin recibe el pulso reflejado y mide el tiempo que tarda en regresar.
- **const int servoPin = 10;**: Define el pin 10 como el pin de control del servomotor. Este pin enviará las señales de control para mover el servomotor.
- **const int ledPin = 13;**: Define el pin 13 como el pin del LED. Se usa para indicar visualmente el estado de la puerta (abierta o cerrada).

Parámetros de control:

- **const int openThreshold = 10;**: Define el umbral de distancia (en centímetros) para abrir la puerta. Si la distancia detectada por el sensor es menor a este valor, se activa la apertura de la puerta.
- **const int openPosition = 90;**: Define la posición del servomotor para cuando la puerta está abierta (90 grados).
- **const int closedPosition = 0;**: Define la posición del servomotor para cuando la puerta está cerrada (0 grados).

```
#include <Servo.h>

const int trigPin = 8;  // Pin de disparo del sensor ultrasónico
const int echoPin = 9;  // Pin de eco del sensor ultrasónico
const int servoPin = 10; // Pin del servomotor
const int ledPin = 13;  // Pin del LED (opcional)

const int openThreshold = 10; // Distancia en cm para abrir la
puerta
const int openPosition = 90; // Posición de la puerta abierta
const int closedPosition = 0; // Posición de la puerta cerrada
```

1. Servo myServo;:

- Crea una instancia de la clase Servo llamada myServo. Esto permite controlar el servomotor utilizando los métodos de la biblioteca Servo.h. Con esta instancia, se podrán enviar señales al servomotor para moverlo a posiciones específicas, como abrir o cerrar la puerta.

2. bool puertaAbierta = false;:

- Declara una variable booleana (bool) llamada puertaAbierta y la inicializa en false. Esta variable se utiliza para rastrear el estado de la puerta, indicando si está abierta (true) o cerrada (false). Esto ayuda a evitar que la puerta se abra o cierre repetidamente cuando no es necesario.

```
Servo myServo;
bool puertaAbierta = false; // Estado de la puerta
```

Este bloque de código es la función setup(), que se ejecuta una vez al inicio del programa. Configura los pines y establece los estados iniciales de los componentes:

1. pinMode(trigPin, OUTPUT);:

- Configura el pin del sensor ultrasónico (trigPin) como salida. Este pin enviará pulsos ultrasónicos.

2. pinMode(echoPin, INPUT);:

- Configura el pin del sensor ultrasónico (echoPin) como entrada. Este pin recibirá los pulsos reflejados para medir la distancia.

3. pinMode(ledPin, OUTPUT);:

- Configura el pin del LED (ledPin) como salida. Esto permite encender y apagar el LED para indicar el estado de la puerta.

4. myServo.attach(servoPin);:

- Asocia el servomotor al pin definido por servoPin. Esto habilita el control del servomotor a través de este pin.

5. myServo.write(closedPosition);:

- Mueve el servomotor a la posición de cerrado (closedPosition), que se ha definido como 0 grados. Esto asegura que la puerta comience cerrada al inicio del programa.

6. digitalWrite(ledPin, LOW);:

- Apaga el LED al inicio. Esto indica que la puerta está cerrada inicialmente.

7. Serial.begin(9600);:

- Inicia la comunicación serial a una velocidad de 9600 baudios. Esto permite enviar datos al monitor serial para depuración o monitoreo, como la distancia medida por el sensor ultrasónico.

```
void setup() {
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(ledPin, OUTPUT);
  myServo.attach(servoPin);
  myServo.write(closedPosition); // Inicialmente, la puerta está
cerrada
  digitalWrite(ledPin, LOW); // LED apagado
  Serial.begin(9600);
}
```

Esta es la función loop(), que se ejecuta continuamente después de setup(). Aquí se controla el funcionamiento del sensor ultrasónico, el servomotor y el LED para abrir y cerrar la puerta según la distancia detectada:

1. Variables Locales:

- **long duration;**: Almacena la duración del pulso reflejado, medida en microsegundos.
- **int distance;**: Almacena la distancia calculada a partir del pulso reflejado.

2. Enviar Pulso al Sensor Ultrasónico:

- **digitalWrite(trigPin, LOW); delayMicroseconds(2);**:

  - Pone el pin de disparo (trigPin) en bajo y espera 2 microsegundos para asegurar que el pulso inicial esté limpio.

- **digitalWrite(trigPin, HIGH); delayMicroseconds(10);**:
    - Envía un pulso de 10 microsegundos al sensor para iniciar la medición.
- **digitalWrite(trigPin, LOW);**:
    - Baja el pin para finalizar el pulso.

3. Leer el Pulso de Retorno del Sensor:

- **duration = pulseIn(echoPin, HIGH);**:
    - Mide el tiempo que tarda el pulso en regresar al sensor. Este tiempo se almacena en la variable duration.
- **distance = (duration / 2) / 29.1;**:
    - Calcula la distancia en centímetros dividiendo la duración por 2 (ida y vuelta) y luego por 29.1 (conversión de tiempo a distancia).

4. Mostrar la Distancia en el Monitor Serial:

- **Serial.print("Distancia: "); Serial.print(distance); Serial.println(" cm");**:
    - Muestra la distancia calculada en el monitor serial para monitoreo y depuración.

5. Control de la Puerta:

- **if (distance < openThreshold && !puertaAbierta)**:
    - Si la distancia es menor que el umbral (openThreshold) y la puerta está cerrada (!puertaAbierta), llama a la función abrirPuerta() y cambia el estado de la puerta a abierta (puertaAbierta = true).
- **else if (distance >= openThreshold && puertaAbierta)**:
    - Si la distancia es mayor o igual al umbral y la puerta está abierta, llama a la función cerrarPuerta() y cambia el estado de la puerta a cerrada (puertaAbierta = false).

6. Delay para Evitar Lecturas Continuas:

- **delay(500);**:

- Espera 500 milisegundos antes de realizar otra medición para evitar lecturas continuas y oscilaciones en el estado de la puerta.

```
void loop() {
  long duration;
  int distance;

  // Enviar pulso al sensor ultrasónico
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // Leer el pulso de retorno del sensor
  duration = pulseIn(echoPin, HIGH);
  distance = (duration / 2) / 29.1; // Convertir tiempo a distancia

  Serial.print("Distancia: ");
  Serial.print(distance);
  Serial.println(" cm");

  if (distance < openThreshold && !puertaAbierta) {
    abrirPuerta();
    puertaAbierta = true;
  } else if (distance >= openThreshold && puertaAbierta) {
    cerrarPuerta();
    puertaAbierta = false;
  }

  delay(500); // Espera para evitar lecturas continuas
}
```

Estas dos **funciones**, abrirPuerta() y cerrarPuerta(), controlan el movimiento del servomotor para abrir y cerrar la puerta, además de encender y apagar un LED como indicador visual.

abrirPuerta()

1. Movimiento del Servomotor para Abrir la Puerta:

- Un bucle for incrementa la posición del servomotor desde closedPosition (posición cerrada) hasta openPosition (posición abierta) en pasos de 1 grado.
- **myServo.write(pos);**: Envía la señal al servomotor para moverse a la posición especificada.

- **delay(15);**: Espera 15 milisegundos para suavizar el movimiento y ajustar la velocidad de apertura.

2. Indicador LED:

- **digitalWrite(ledPin, HIGH);**: Enciende el LED, indicando que la puerta está abierta.

3. Mensaje en el Monitor Serial:

- **Serial.println("Puerta abierta.");**: Muestra el mensaje en el monitor serial para confirmar que la puerta se ha abierto.

cerrarPuerta()

1. Movimiento del Servomotor para Cerrar la Puerta:

- Un bucle for decrementa la posición del servomotor desde openPosition hasta closedPosition en pasos de 1 grado.
- **myServo.write(pos);**: Envía la señal al servomotor para moverse a la posición especificada.
- **delay(15);**: Espera 15 milisegundos para ajustar la velocidad de cierre.

2. Indicador LED:

- **digitalWrite(ledPin, LOW);**: Apaga el LED, indicando que la puerta está cerrada.

3. Mensaje en el Monitor Serial:

- **Serial.println("Puerta cerrada.");**: Muestra el mensaje en el monitor serial para confirmar que la puerta se ha cerrado.

Resumen

- **abrirPuerta()**: Abre la puerta gradualmente, enciende el LED, y muestra un mensaje en el monitor serial.
- **cerrarPuerta()**: Cierra la puerta gradualmente, apaga el LED, y muestra un mensaje en el monitor serial.

```
void abrirPuerta() {
  for (int pos = closedPosition; pos <= openPosition; pos += 1) {
    myServo.write(pos);
    delay(15); // Ajustar velocidad de apertura
  }
  digitalWrite(ledPin, HIGH); // Enciende el LED
  Serial.println("Puerta abierta.");
}

void cerrarPuerta() {
  for (int pos = openPosition; pos >= closedPosition; pos -= 1) {
    myServo.write(pos);
    delay(15); // Ajustar velocidad de cierre
  }
  digitalWrite(ledPin, LOW); // Apaga el LED
  Serial.println("Puerta cerrada.");
}
```

Conexiones de los componentes electrónicos:

Una vez que tenemos listo nuestro código, es importante conectar correctamente nuestros componentes, ya que, si lo hacemos mal, el código no funcionará como debería.

Vamos a seguir los mismos pasos que hemos estado usando para las conexiones.

Ultrasonic1: GND conectado a uno: GND2
Ultrasonic1: ECHO conectado a uno:9
Ultrasonic1: TRIG conectado a uno:8
Ultrasonic1: VCC conectado a uno:5V
LED1:C conectado a uno: AREF
LED1: A conectado a r1:1
r1:2 conectado a uno:13
servo1: GND conectado a uno: GND2
servo1: V+ conectado a uno: 5V
servo1: PWM conectado a uno:10

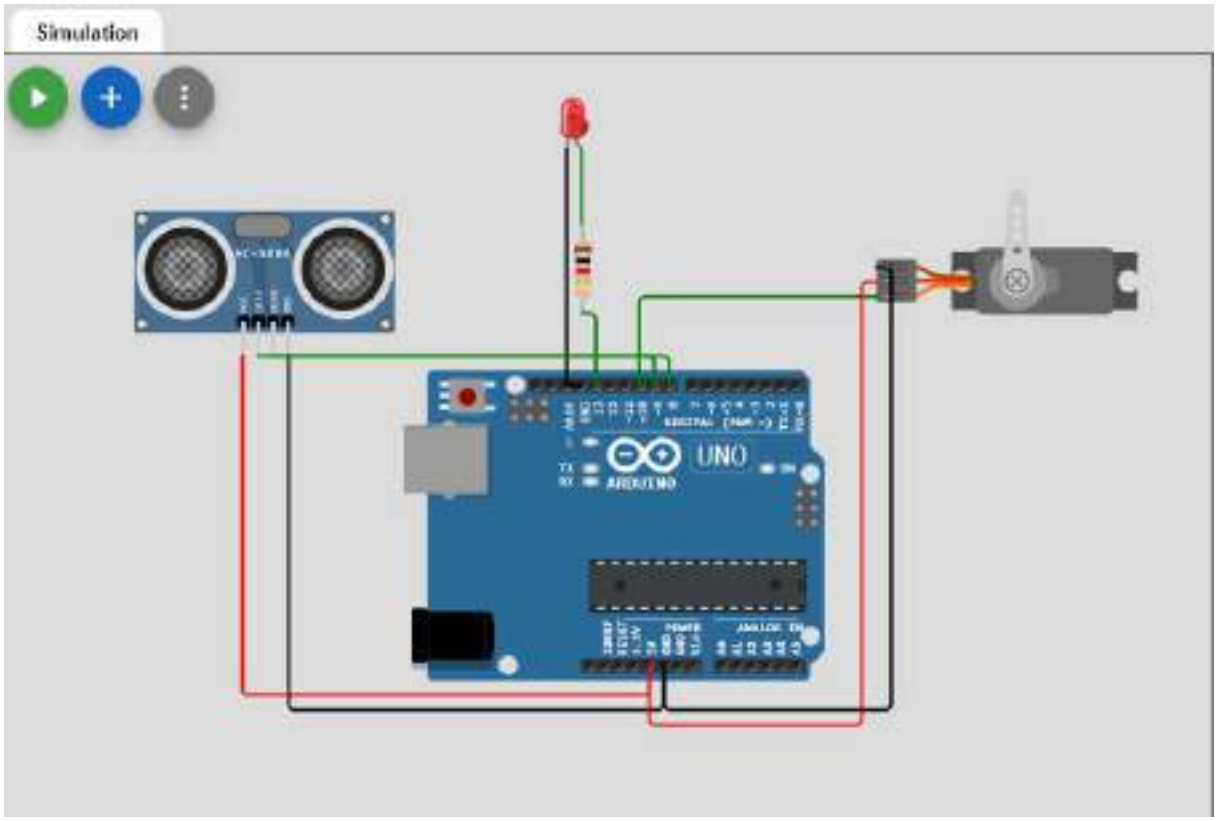


Figura 1: circuito correctamente conectado como lo muestran las indicaciones

Prueba final:

Has llegado a la parte final de esta práctica. Ahora solo debes compilar y ejecutar tu proyecto, y verificar que todo funcione como debe.

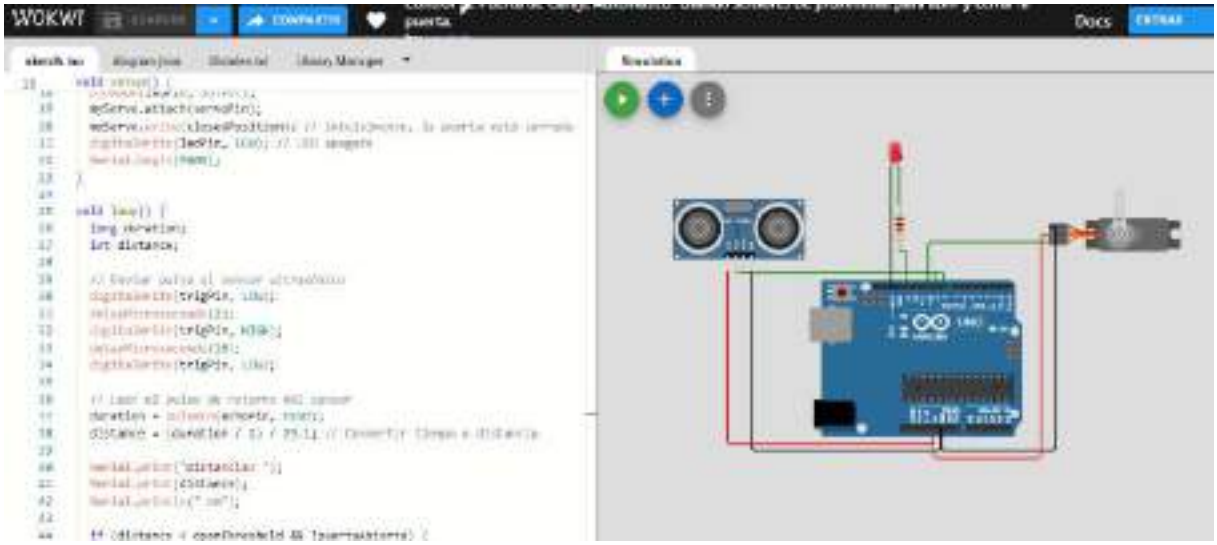


Figura 2: código y circuito listo para su simulación

Ejecutar nuestro proyecto

Cuando hagas clic en el ícono de ejecutar, solo tendrás que ajustar la distancia de nuestro sensor ultrasónico. Al estar a una distancia de 2 cm, la puerta se abrirá automáticamente, y el LED se encenderá para indicarlo. Si luego mueves el sensor a una distancia de 401 cm, la puerta se cerrará, y el LED se apagará, mostrando claramente el cambio de estado.

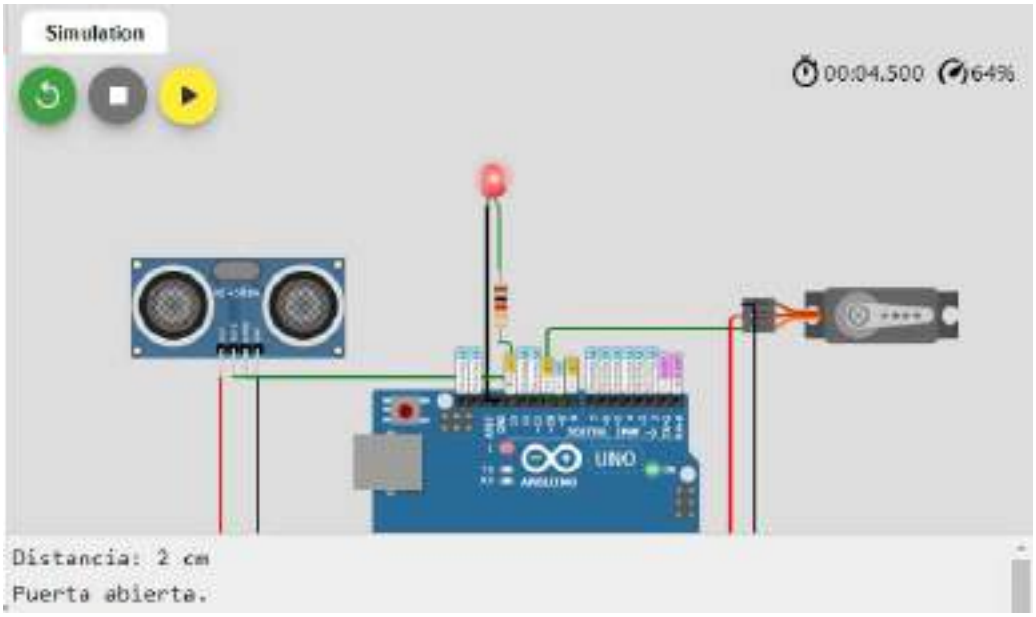


Figura 3: Al ejecutar el código, el sensor de distancia detectará una distancia de 2 cm. En respuesta, el sistema mostrará el mensaje “Puerta abierta” en el monitor serial, encenderá el LED como indicador y abrirá la puerta simulada moviendo el servomotor

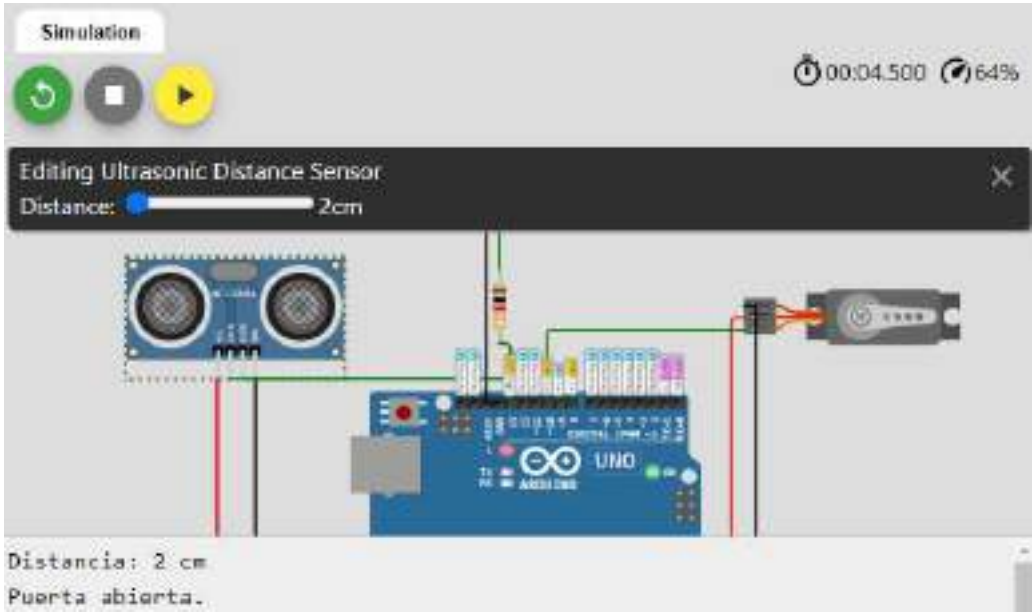


Figura 4: Al ejecutar el código y presionar el sensor de distancia, podrás manipular la simulación de proximidad de objetos

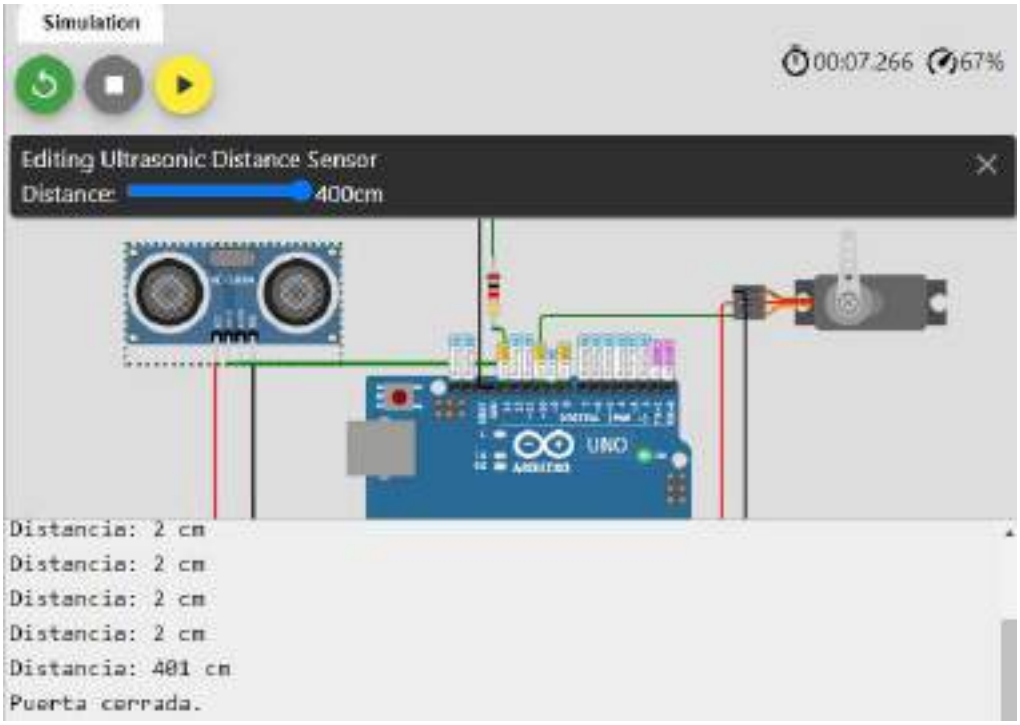


Figura 5: Al presionar el sensor de distancia, podrás manipular la simulación de proximidad de objetos. Si mueves el sensor a 400 cm, no detectará ninguna proximidad y, en respuesta, el sistema procederá a cerrar la puerta simulada

PRÁCTICAS DE SISTEMAS ELECTRÓNICOS

SISTEMA DE ILUMINACIÓN INTELIGENTE CON DETECCIÓN DE MOVIMIENTO: USANDO UN SENSOR PIR Y CONTROL DE LUZ BASADO EN LA PRESENCIA DE PERSONAS

El “Sistema de Iluminación Inteligente con Detección de Movimiento” es una solución moderna diseñada para optimizar el uso de la luz en espacios residenciales, comerciales o industriales. Utiliza un sensor PIR (Infrarrojo Pasivo) para detectar la presencia de personas en una habitación o área específica. Cuando se detecta movimiento, el sistema activa automáticamente las luces, proporcionando iluminación solo cuando es necesario. Esto no solo mejora la comodidad y seguridad, sino que también contribuye a un uso más eficiente de la energía.

La principal ventaja de este sistema radica en su capacidad de ajustar el encendido y apagado de las luces según la actividad en el entorno. Por ejemplo, en pasillos, baños, o espacios comunes, la luz se enciende cuando alguien entra y se apaga poco tiempo después de que la persona se va. Esto reduce significativamente el desperdicio de energía en comparación con los sistemas de iluminación tradicionales que dependen de la intervención manual para su funcionamiento.

Además de su eficiencia energética, el sistema de iluminación inteligente también puede integrarse con otras tecnologías, como temporizadores, niveles de brillo ajustables y sistemas de automatización del hogar. Esto permite crear un entorno de iluminación adaptable y personalizado para diferentes situaciones. La combinación de la detección de movimiento y el control inteligente de la luz convierte a este sistema en una opción ideal para hogares y edificios inteligentes, promoviendo así la sostenibilidad y el confort.

DISEÑO DEL PROTOTIPO

Para el diseño de este prototipo usaremos la plataforma **Wokwi.com**

Descripción del Prototipo:

El fin de este prototipo es que cuando el sensor PIR detecta movimiento, el LED se enciende y el código imprime “Movimiento detectado”. Cuando no hay movimiento, el LED se apaga y el código muestra “Sin movimiento”.

Componentes Utilizados:

Componentes	Cantidad	Especificaciones
ESP32	1	Microcontrolador principal que gestiona el sistema.
Sensor de movimiento	1	Un sensor de movimiento detecta la presencia o movimiento en un área, activando dispositivos como luces o alarmas cuando se registra actividad.
LED	1	Es un componente electrónico que emite luz cuando pasa una corriente eléctrica a través de él.
Resistencia 229 ohm	1	Una resistencia de 229 ohmios limita la corriente eléctrica en un circuito para proteger los componentes.

Simulación del Comportamiento:

Sensor de Movimiento (PIR): Detecta la presencia o movimiento en su campo de acción. En el código, si el sensor detecta movimiento (pirState == HIGH), se activa el LED.

LED: Se enciende (digitalWrite(ledPin, HIGH)) cuando el sensor PIR detecta movimiento, proporcionando una indicación visual. Se apaga (digitalWrite(ledPin, LOW)) cuando no se detecta movimiento.

Resistencia de 229 ohmios: Está conectada en serie con el LED para limitar la corriente que pasa a través de él, evitando daños y asegurando un funcionamiento correcto del LED.

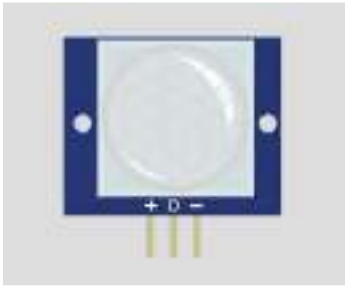
Pasos para la elaboración de la práctica:

vamos a preparar nuestros componentes para nustreo proyecto para eso usaremos la plataforma wokwi.

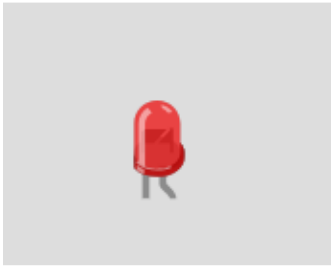
Una vez que estemos en la plataforma Wokwi.com, vamos a escoger los siguientes componentes:

- ESP32

- Sensor de movimiento (PIR)



- LED



- Resistencia 229 ohm

Una vez que tengamos los componentes, estaremos listos para comenzar a trabajar.

Desarrollo del código:

Ahora vamos a desarrollar el código de nuestro proyecto, el cual dará las instrucciones que debe seguir nuestro prototipo.

Asegúrate de seguir cada paso con atención y en el orden indicado. Una conexión incorrecta o un código mal escrito puede hacer que el proyecto no funcione como se espera. Tómate tu tiempo para revisar las conexiones y el código antes de realizar la simulación.

Código

- **const int ledPin = 13;**: Define el pin 13 como el pin para conectar el LED. El LED se encenderá o apagará basado en las lecturas del sensor PIR.
- **const int pirPin = 14;**: Define el pin 14 como el pin para conectar el sensor PIR. Este sensor detecta movimiento y envía una señal al pin 14 para activar o desactivar el LED.

```
const int ledPin = 13; // Pin del LED
const int pirPin = 14; // Pin del sensor PIR
```

Función setup():

- **Serial.begin(115200);**: Inicia la comunicación serial a una velocidad de 115200 baudios, permitiendo la visualización de datos en el monitor serial.

- **pinMode(ledPin, OUTPUT);**: Configura el pin del LED (pin 13) como salida, permitiendo encender y apagar el LED.
- **pinMode(pirPin, INPUT);**: Configura el pin del sensor PIR (pin 14) como entrada, permitiendo leer las señales del sensor para detectar movimiento.

```
void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  pinMode(pirPin, INPUT);
}
```

Función loop():

- **int pirState = digitalRead(pirPin);**: Lee el estado del sensor PIR conectado al pin 14. pirState será HIGH si se detecta movimiento y LOW si no se detecta movimiento.
- **if (pirState == HIGH) { ... }**: Verifica si el sensor PIR detecta movimiento:
- **Serial.println("Movimiento detectado");**: Imprime "Movimiento detectado" en el monitor serial.
- **digitalWrite(ledPin, HIGH);**: Enciende el LED conectado al pin 13.
- **else { ... }**: Si no se detecta movimiento:
- **Serial.println("Sin movimiento");**: Imprime "Sin movimiento" en el monitor serial.
- **digitalWrite(ledPin, LOW);**: Apaga el LED.
- **delay(100);**: Pausa la ejecución del código durante 100 milisegundos para evitar lecturas erráticas y permitir una detección estable.

```
void loop() {
  int pirState = digitalRead(pirPin);

  if (pirState == HIGH) {  // Si se detecta movimiento
    Serial.println("Movimiento detectado");
    digitalWrite(ledPin, HIGH); // Enciende la luz
  } else {
    Serial.println("Sin movimiento");
    digitalWrite(ledPin, LOW); // Apaga la luz
  }

  delay(100); // Espera corta para evitar lecturas erráticas
}
```

Conexiones de los componentes electrónicos:

Una vez que tengamos el código listo, es esencial conectar correctamente los componentes, ya que una conexión incorrecta hará que el código no funcione como debería. Sigamos los mismos pasos que hemos utilizado para las conexiones.

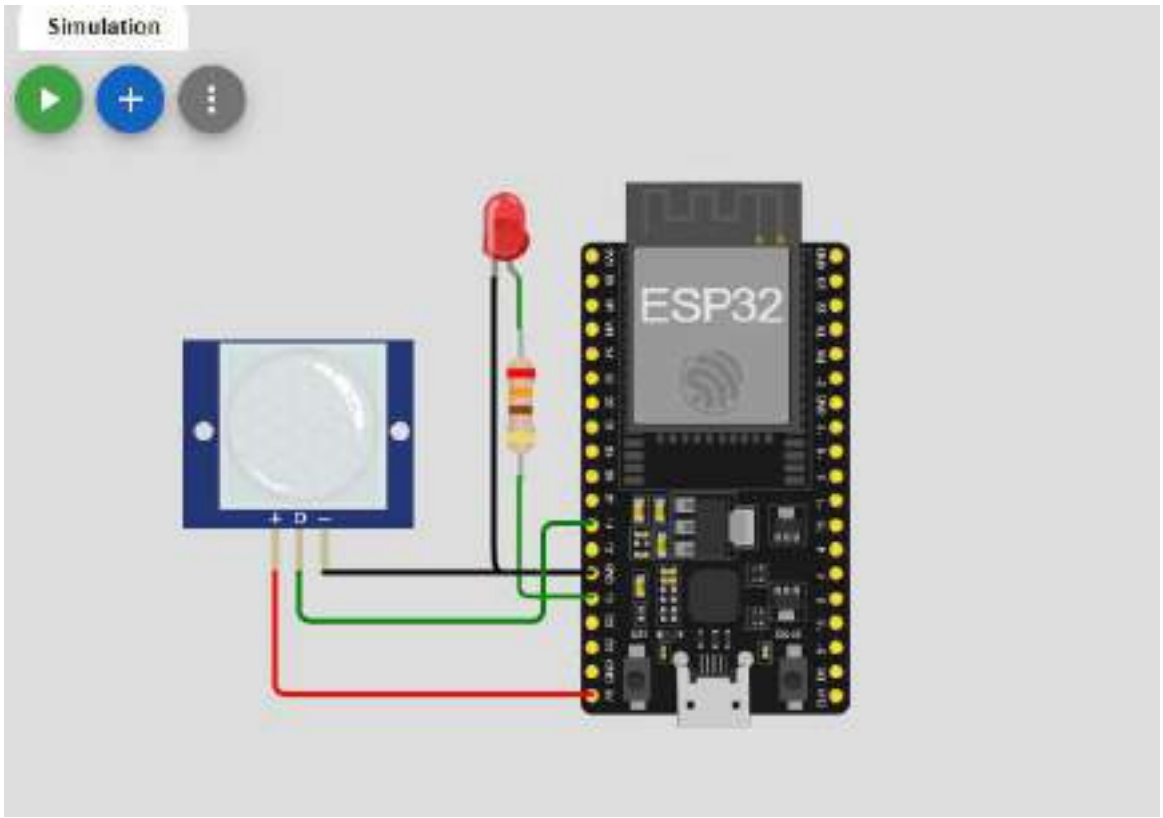


Figura 6: circuito conectado correctamente como lo dice en las indicaciones

Prueba final:

Has llegado a la parte final de esta práctica. Ahora solo debes compilar y ejecutar tu proyecto, y verificar que todo funcione como debe.

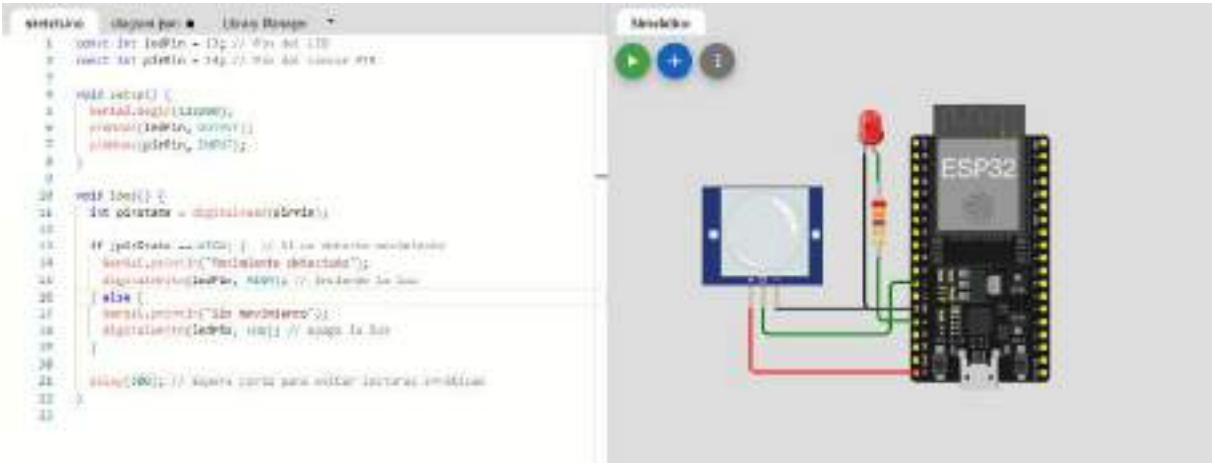


Figura 7: código y circuito listos para simular

Ejecutar nuestro proyecto

Cuando hagas clic en el ícono de ejecutar, presiona en el sensor PIR y selecciona “Simulate Motion” para simular el movimiento. Si se detecta movimiento, el sistema imprimirá “Movimiento detectado” en el monitor serial y el LED se encenderá. Mientras no haya movimiento, se imprimirá “Sin movimiento” en el monitor serial y el LED permanecerá apagado.

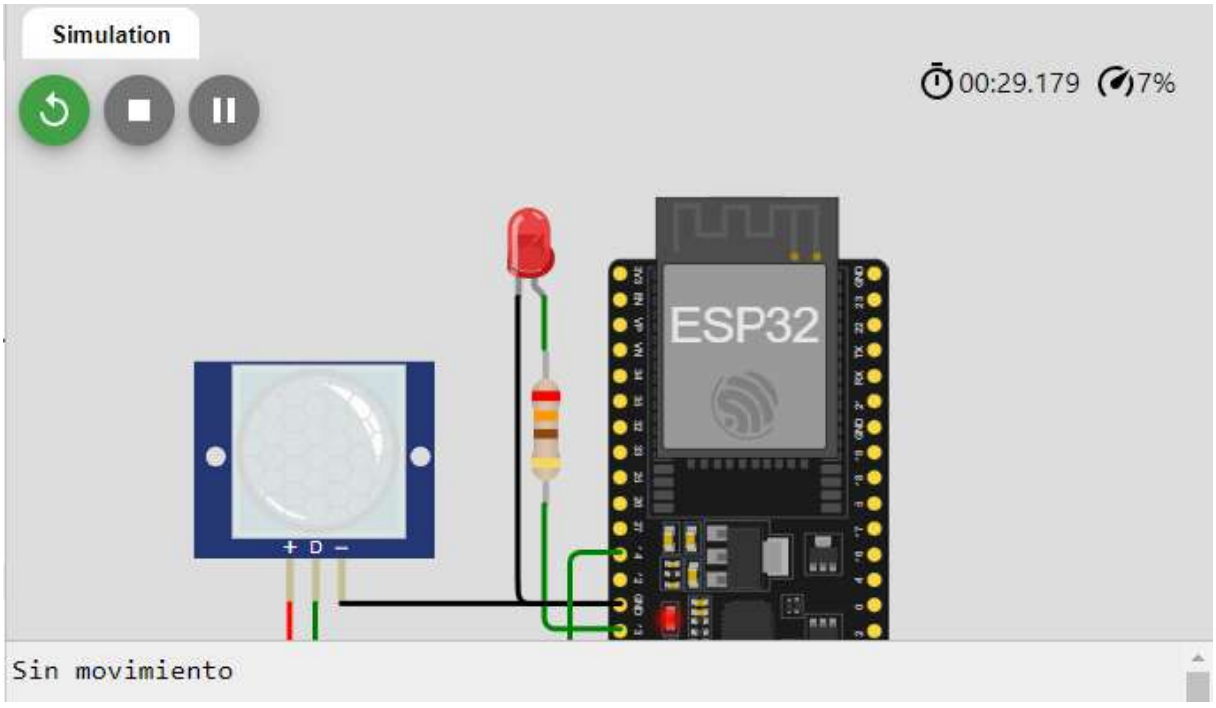


Figura 8: Al ejecutar el código se muestra un mensaje de que no existe movimiento

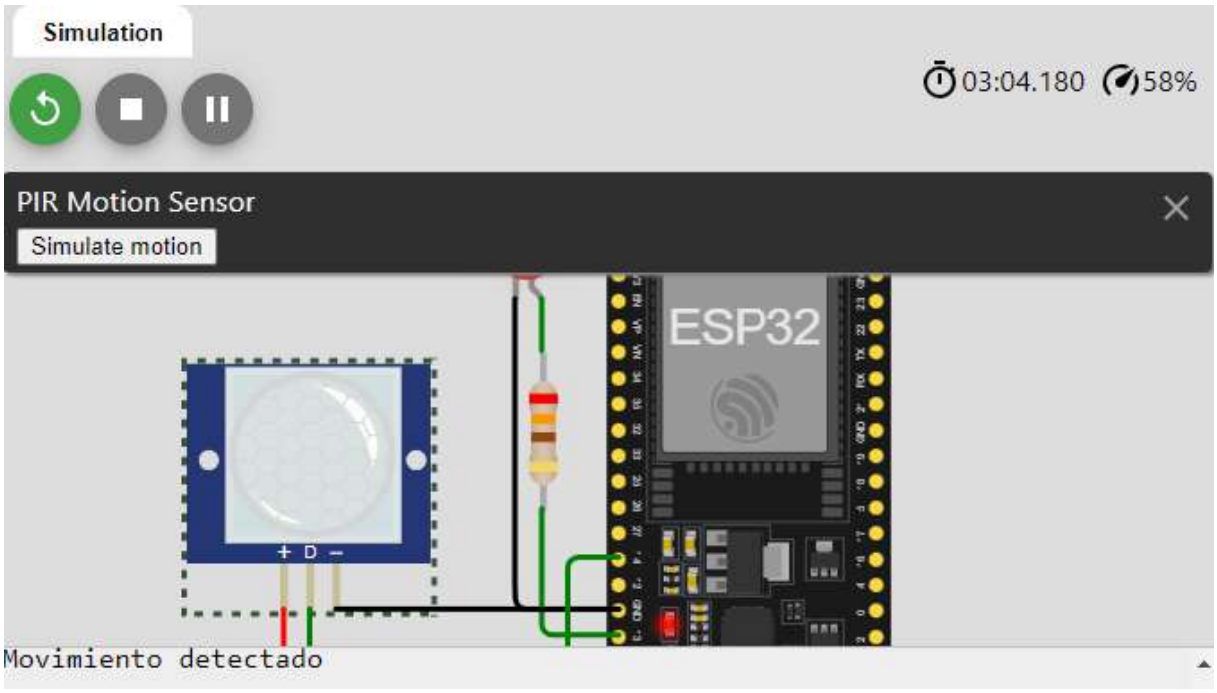


Figura 9: Al ejecutar el código y presionar el sensor PIR,
se simulará el movimiento y se mostrará el mensaje “Movimiento detectado”

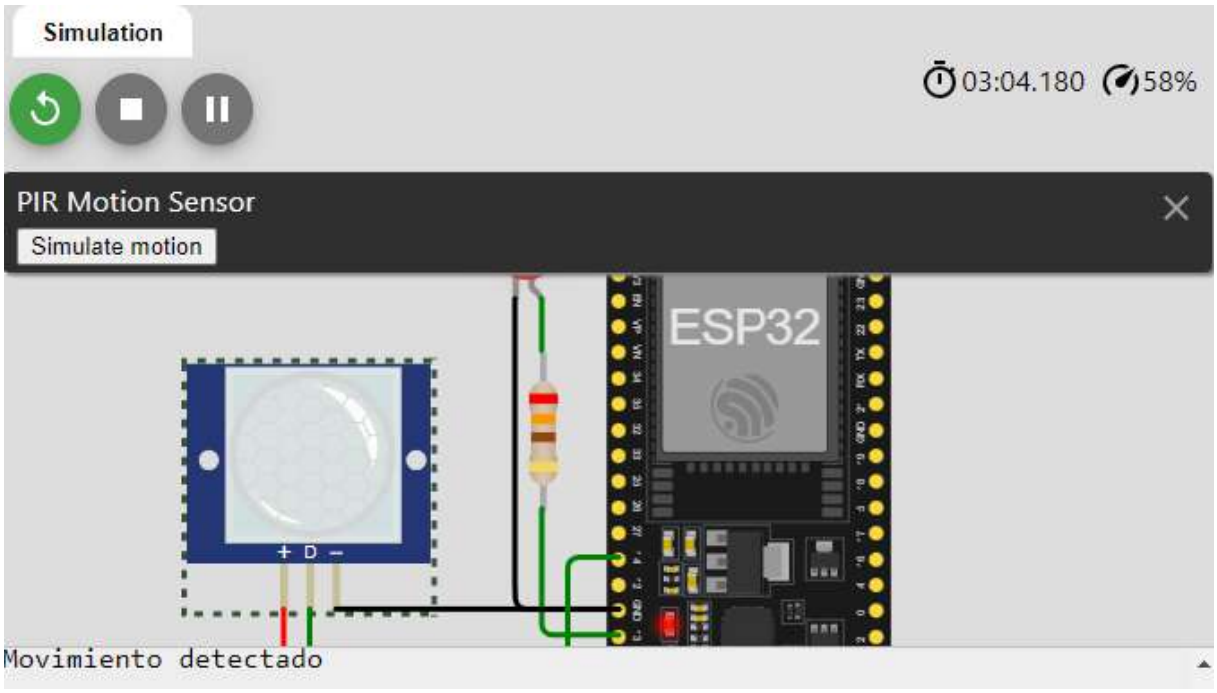


Figura 10: se simulará el movimiento, se mostrará el mensaje “Movimiento detectado”
y el LED se encenderá para indicar la detección de movimiento

PRÁCTICAS DE SISTEMAS ELECTRÓNICOS

CONTROL AUTOMÁTICO DE PERSIANAS INTELIGENTES CON LDR: BASADO EN LA DETECCIÓN DE LUZ

El “Control Automático de Persianas Inteligentes con LDR” es un sistema innovador diseñado para ajustar las persianas de un espacio en función de los niveles de luz ambiental. Utilizando un sensor de luz, conocido como LDR (Light Dependent Resistor), este sistema automatiza la apertura y cierre de las persianas para mantener el ambiente en condiciones óptimas de iluminación. Esto no solo proporciona un mayor confort en el hogar o la oficina, sino que también ayuda a regular la temperatura interior, reduciendo la necesidad de aire acondicionado o calefacción.

El funcionamiento del sistema se basa en la medición constante de la intensidad de la luz que entra en la habitación. Cuando el sensor LDR detecta niveles de luz que superan un umbral predefinido, las persianas se cierran para reducir el deslumbramiento y el calentamiento excesivo. En contraste, si los niveles de luz son bajos, las persianas se abren para permitir una mayor entrada de luz natural, creando un entorno más agradable y eficiente energéticamente.

Este sistema no solo mejora la eficiencia energética al optimizar el uso de la luz natural, sino que también contribuye al ahorro en las facturas de electricidad y aumenta la vida útil de las persianas al reducir el desgaste mecánico. Con la integración de tecnologías modernas y un diseño orientado al usuario, el control automático de persianas inteligentes con LDR ofrece una solución práctica y avanzada para la gestión de la luz en interiores.

DISEÑO DEL PROTOTIPO

Para el diseño de este prototipo usaremos la plataforma **Wokwi.com**

Descripción del Prototipo:

El fin de este prototipo es que cuando el sensor PIR detecta movimiento, el LED se enciende y el código imprime “Movimiento detectado”. Cuando no hay movimiento, el LED se apaga y el código muestra “Sin movimiento”.

Componentes Utilizados:

Componentes	Cantidad	Especificaciones
Arduino uno	1	Microcontrolador principal que gestiona el sistema.
Servomotor	1	Simula la apertura y cierre de unas persianas.
fotorresistor (LDR) sensor module.	1	Un módulo sensor de fotorresistor (LDR) mide la luz ajustando su resistencia según la intensidad luminosa.

Simulación del Comportamiento:

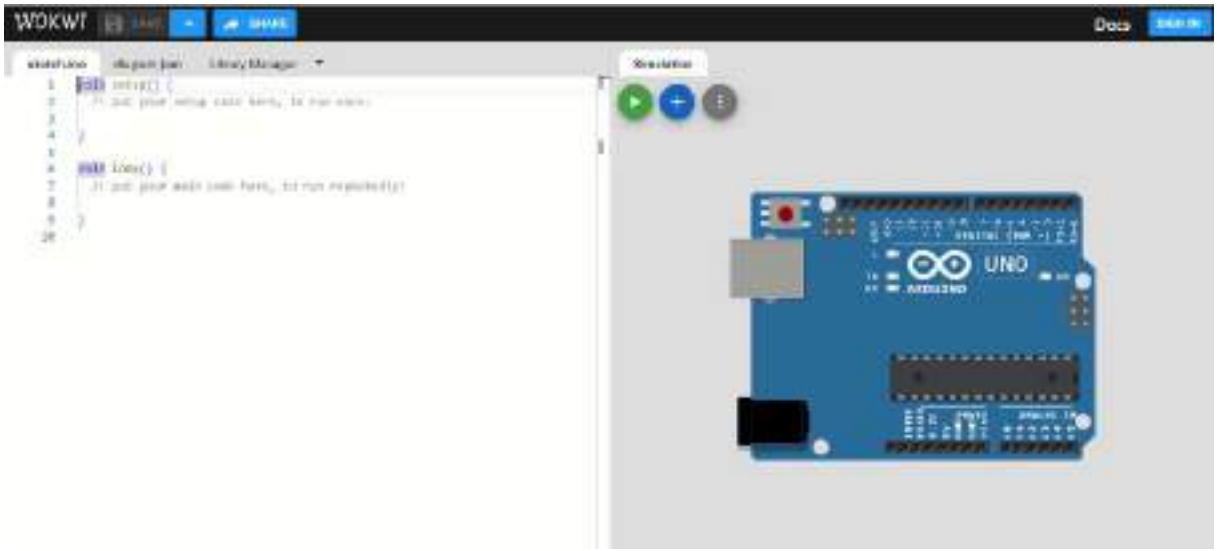
Servomotor: Controla la apertura y cierre de las persianas. Se mueve de 0° a 180° para abrir las persianas y de 180° a 0° para cerrarlas, ajustando la posición del servomotor basado en la lectura del fotoresistor.

Fotoresistor (LDR): Mide la intensidad de la luz. Si la luz detectada es menor que el umbral el servomotor abre las persianas; si la luz es igual o mayor que el umbral, el servomotor cierra las persianas.

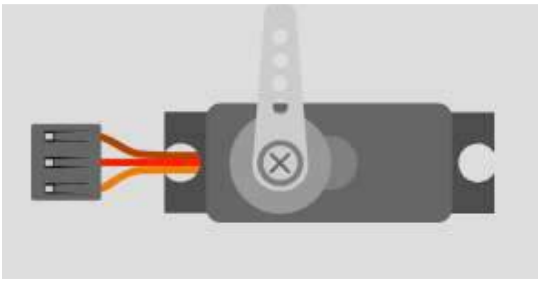
Pasos para la elaboración de la practica:

Vamos a preparar los componentes para nuestro proyecto utilizando la plataforma Wokwi. Primero, accede a Wokwi.com y selecciona los siguientes componentes:

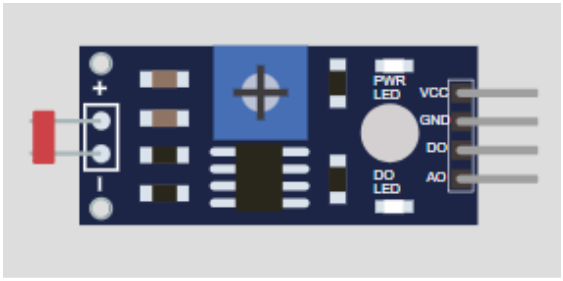
- Arduino uno



- Servomotor



- fotorresistor (LDR) sensor module.



Una vez que tengamos los componentes, estaremos listos para comenzar a trabajar.

Desarrollo del código:

Ahora vamos a desarrollar el código de nuestro proyecto, el cual dará las instrucciones que debe seguir nuestro prototipo.

Asegúrate de seguir cada paso con atención y en el orden indicado. Una conexión incorrecta o un código mal escrito puede hacer que el proyecto no funcione como se espera. Tómate tu tiempo para revisar las conexiones y el código antes de realizar la simulación.

Código

- **#include <Servo.h>**: Incluye la biblioteca de control de servomotores.
- **const int ldrPin = A0;**: Define el pin analógico donde está conectado el fotoresistor (LDR).
- **const int servoPin = 9;**: Define el pin digital al que está conectado el servomotor.
- **const int lightThreshold = 500;**: Establece el umbral de luz para decidir cuándo abrir o cerrar las persianas. Ajusta este valor según tus necesidades.
- **Servo myServo;**: Crea una instancia del objeto Servo para controlar el servomotor.
- **enum State { OPEN, CLOSED };**: Define un enumerador para los estados de la persiana: abierta (OPEN) o cerrada (CLOSED).
- **State currentState = CLOSED;**: Inicializa el estado de la persiana en cerrado.

```
#include <Servo.h>

const int ldrPin = A0;  // Pin del fotoresistor (salida analógica)
const int servoPin = 9; // Pin del servomotor

const int lightThreshold = 500; // Umbral de luz para controlar
las persianas (ajustar según necesidad)
Servo myServo;

enum State { OPEN, CLOSED };
State currentState = CLOSED; // Estado inicial de la persiana
```

Función setup():

- **myServo.attach(servoPin);**: Conecta el servomotor al pin especificado (servoPin).
- **Serial.begin(9600);**: Inicializa la comunicación serial a una velocidad de 9600 baudios, permitiendo la transmisión de datos al monitor serial.
- **myServo.write(0);**: Coloca el servomotor en la posición inicial de 0°, lo que significa que las persianas están cerradas al inicio del programa.

```
void setup() {
  myServo.attach(servoPin);
  Serial.begin(9600);
  myServo.write(0); // Inicialmente en posición cerrada
}
```

Función loop():

- **int ldrValue = analogRead(ldrPin);**: Lee el valor analógico del fotoresistor (LDR) desde el pin ldrPin.
- **Serial.print("Luz detectada: ");**: Muestra el mensaje "Luz detectada:" en el monitor serial.
- **Serial.println(ldrValue);**: Imprime el valor leído del fotoresistor en el monitor serial.
- **if (ldrValue < lightThreshold && currentState == CLOSED)**: Si el valor del LDR es menor que el umbral (lightThreshold) y la persiana está cerrada (currentState == CLOSED), llama a la función abrirPersiana() y cambia el estado a OPEN.
- **else if (ldrValue >= lightThreshold && currentState == OPEN)**: Si el valor del LDR es igual o mayor que el umbral y la persiana está abierta (currentState == OPEN), llama a la función cerrarPersiana() y cambia el estado a CLOSED.
- **delay(1000);**: Espera 1 segundo antes de realizar la siguiente lectura del LDR.

```
void loop() {
  int ldrValue = analogRead(ldrPin); // Lee el valor del
fotoresistor

  Serial.print("Luz detectada: ");
  Serial.println(ldrValue);

  // Control de la persiana basado en la lectura del fotoresistor
  if (ldrValue < lightThreshold && currentState == CLOSED) {
    abrirPersiana();
    currentState = OPEN;
  }
  else if (ldrValue >= lightThreshold && currentState == OPEN) {
    cerrarPersiana();
    currentState = CLOSED;
  }

  delay(1000); // Espera 1 segundo antes de la siguiente lectura
}
```

Funciones abrirPersiana() y cerrarPersiana():

- **void abrirPersiana()**:
- **for (int pos = 0; pos <= 180; pos += 1)**: Un bucle que incrementa la posición del servomotor de 0° a 180°, abriendo las persianas gradualmente.
- **myServo.write(pos);**: Establece la posición del servomotor en el valor actual del bucle.

- **delay(15);**: Introduce una pausa de 15 milisegundos entre cada ajuste de posición, controlando la velocidad de apertura.
- **Serial.println("Persiana abierta.");**: Imprime "Persiana abierta." en el monitor serial cuando la persiana está completamente abierta.
- **void cerrarPersiana()**:
- **for (int pos = 180; pos >= 0; pos -= 1)**: Un bucle que decrementa la posición del servomotor de 180° a 0°, cerrando las persianas gradualmente.
- **myServo.write(pos);**: Establece la posición del servomotor en el valor actual del bucle.
- **delay(15);**: Introduce una pausa de 15 milisegundos entre cada ajuste de posición, controlando la velocidad de cierre.
- **Serial.println("Persiana cerrada.");**: Imprime "Persiana cerrada." en el monitor serial cuando la persiana está completamente cerrada.

Conexiones de los componentes electrónicos:

Una vez que tengamos el código listo, es crucial conectar correctamente los componentes, ya que una conexión incorrecta impedirá que el código funcione correctamente. Sigamos los mismos pasos que hemos utilizado para las conexiones.

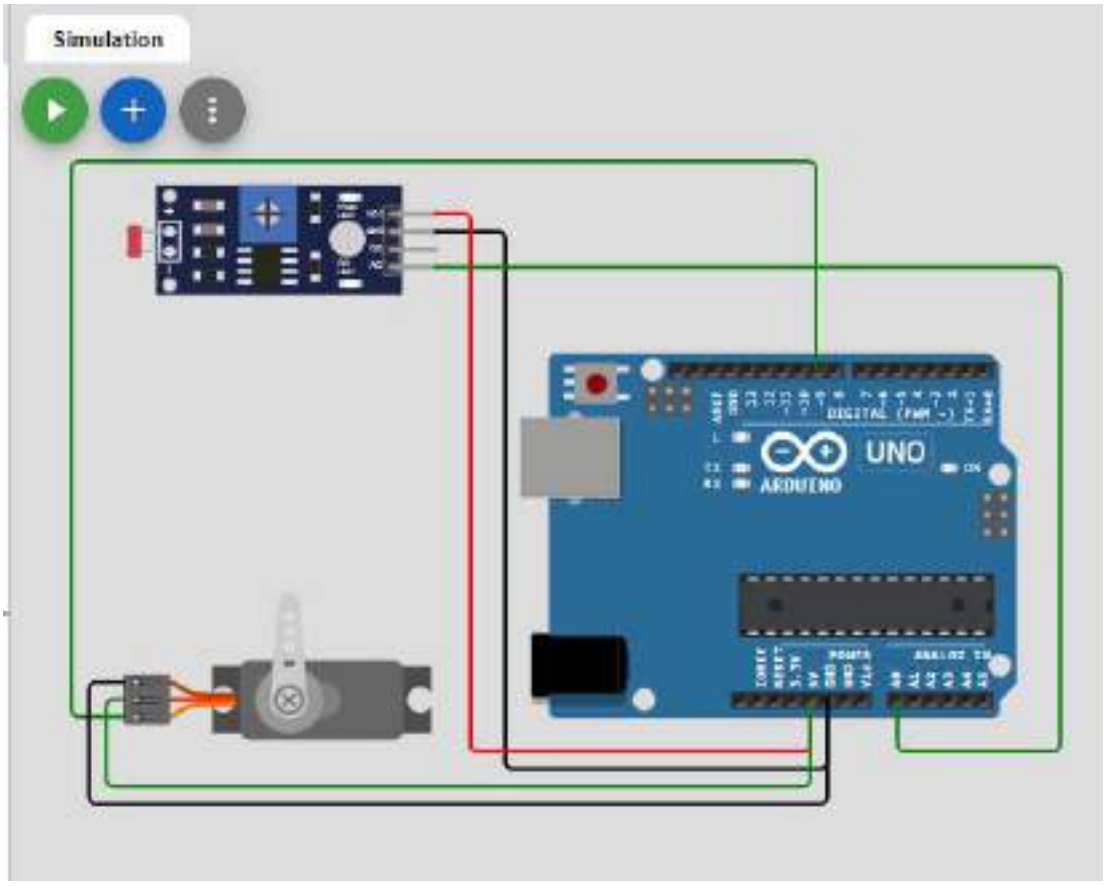


Figura 11: Circuito correctamente conectado según las indicaciones

Prueba final:

Has llegado a la parte final de esta práctica. Ahora solo necesitas compilar y ejecutar tu proyecto, y verificar que todo funcione correctamente.

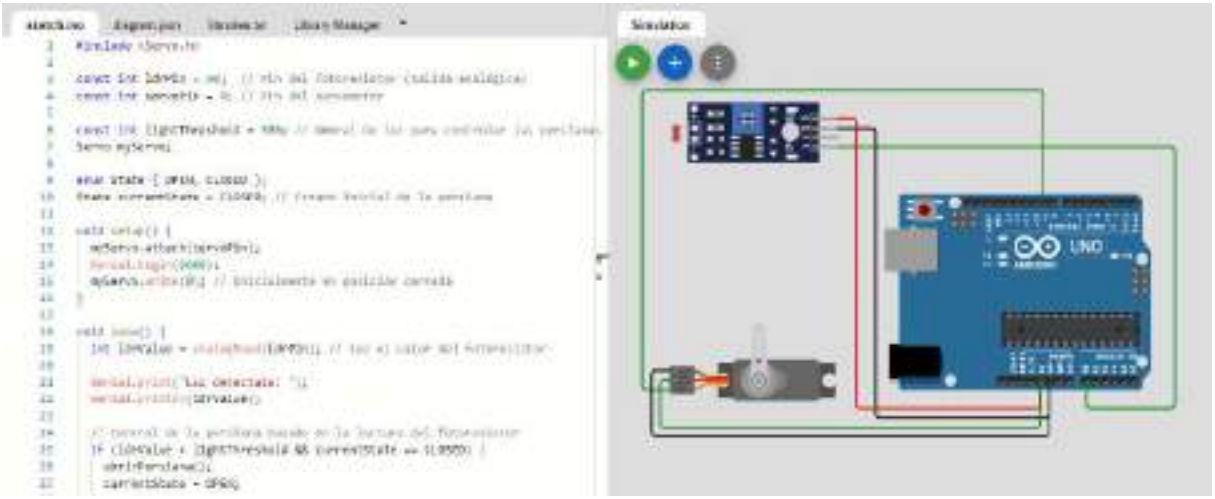


Figura 12: código y circuito listos para simular

Ejecutar nuestro proyecto

Cuando hagas clic en el ícono de ejecutar, el sistema detectará la luz con el sensor LDR y abrirá las persianas moviendo el servomotor. Al presionar el sensor LDR y cambiar los valores, el sistema detectará la luz y cerrará las persianas moviendo nuevamente el servomotor.

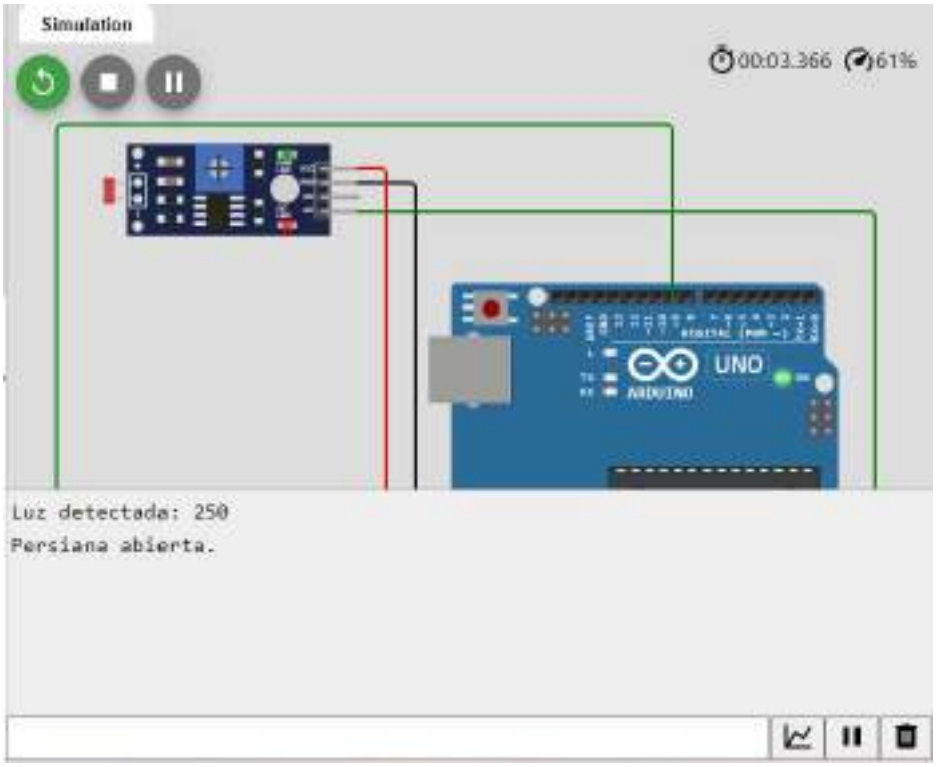


Figura 13: simulación en la que detecta la luz y abre la persiana

Figura 14: simulación en la cual se a ajustado la luz del sensor y la persiana está abierta

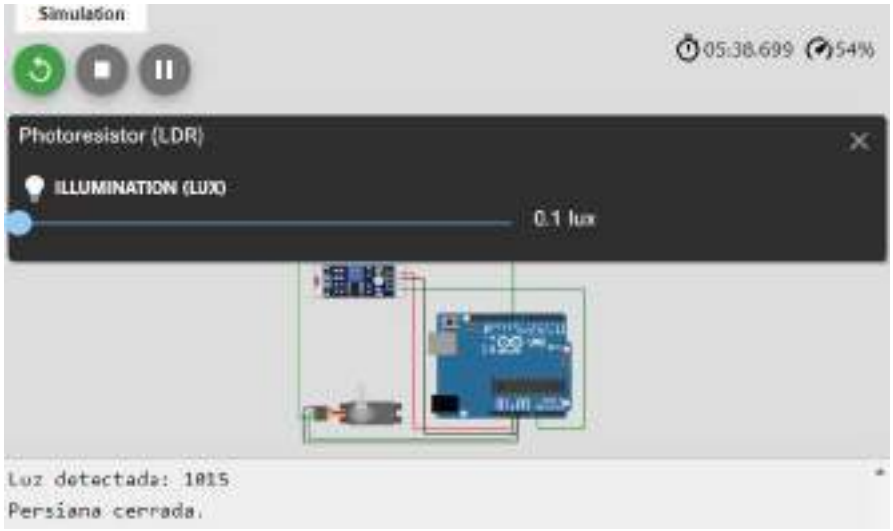


Figura 15: simulación en la cual se a ajustado la luz del sensor cambiando los valores y se cierra la persiana

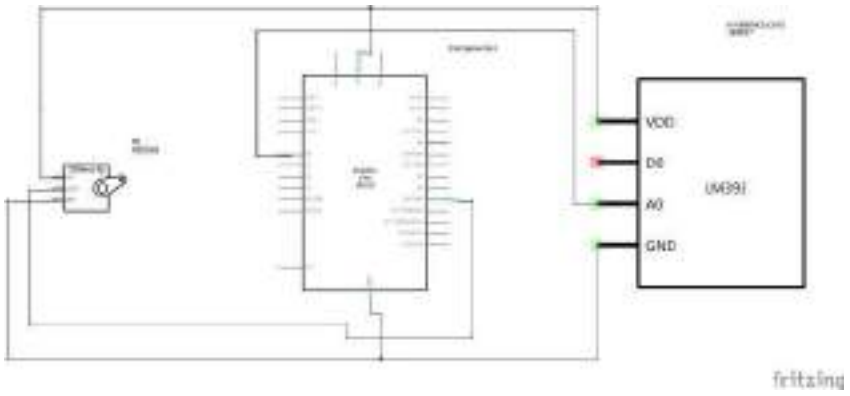


Figura 16: diagrama esquematico de nuestro prototipo

Bibliografía

Cardenas, W. V. (2023.). Método para la detecciún de suplantación de identidad en imágenes digitales del rostro. *Tecnológico Nacional de México*.

Fontalba Roca, M. (2024). SISITEMA EMPOTRADO DE CONTROL DE ACCESO AL USO DE ACTIVOS MEDIANTE RECONOCIMIENTO DE CARAS. *UNIVERSITAT POLITÈCNICA DE VALÈNCIA*.

BIBLIOGRAPHY Brown, S., & Vranesic, Z. (2006). Fundamentos de lógica digital con diseño VHDL (Segunda ed.). McGraw-Hill.

Deschamps, J.-P., Valderrama, E., & Terés, L. (15 de Septiembre de 2017). Digital Systems from logc gates to processors. Springer.

Floyd, T. (2006). Fundamentos de sistemas digitales (Novena ed.). Pearson Educación.

Machado, F., & Borromero, S. (2010). Diseño de circuitos digitales con VHDL (Primera ed.). Universidad Rey Juan Carlos.

Morris Mano, M. (2003). Diseño Digital (Tercera ed.). Pearson Educación.

UNIDAD 04

Diseño Lógico Secuencial

Diseño Lógico Secuencial

TEMA 4.1: INTRODUCCIÓN AL DISEÑO LÓGICO SECUENCIAL

Hasta ahora hemos estudiado circuitos combinacionales: sus salidas dependen exclusivamente de las entradas actuales. Aunque es probable que todos los sistemas digitales tengan circuitos combinacionales, casi todos los que se usan en la práctica también incluyen elementos de almacenamiento, que requieren que el sistema se describa en términos de **lógica secuencial**. Los circuitos **secuenciales** usan elementos de almacenamiento además de compuertas lógicas, y sus salidas son función de las entradas y del estado de los elementos de almacenamiento.

En la figura a continuación se presenta un diagrama de bloques de un circuito secuencial. Consiste en un circuito combinacional al que se conectan elementos de almacenamiento para formar una **trayectoria de retroalimentación**. Los elementos de almacenamiento son dispositivos capaces de guardar información binaria. La información almacenada en estos elementos en cualquier momento dado define el estado del circuito secuencial en ese momento. El circuito secuencial recibe información binaria de entradas externas. Esas entradas, junto con el estado actual de los elementos de almacenamiento, determinan el valor binario de las salidas. También determinan la condición para cambiar el estado de los elementos de almacenamiento.

Diagrama de bloques de un circuito secuencial

El diagrama de bloques indica que las salidas de un circuito secuencial son función no sólo de las entradas, sino también del estado actual de los elementos de almacenamiento.

RECUERDA QUE

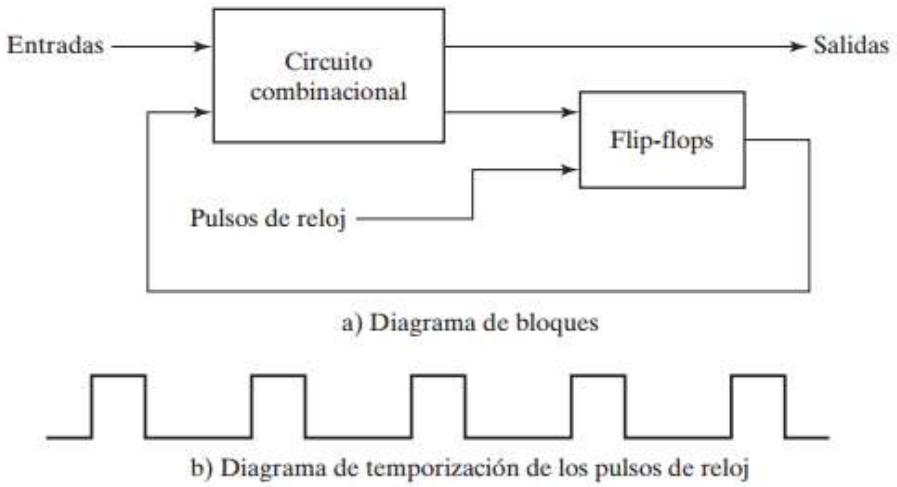
Una trayectoria de retroalimentación es una conexión de la salida de una compuerta a la entrada de otra compuerta, que a su vez forma parte de la entrada a la primera compuerta mencionada.

Hay dos tipos principales de circuitos secuenciales, y su clasificación depende de los tiempos de sus señales.

- Un circuito secuencial **sincrónico** es un sistema cuyo comportamiento se define conociendo sus señales en instantes discretos.
- El comportamiento de un circuito secuencial **asincrónico** depende de las señales de entrada en cualquier instante dado y del orden en que cambian las entradas.

Un circuito secuencial **sincrónico** utiliza señales que afectan a los elementos de almacenamiento únicamente en instantes discretos. La sincronización se logra con un dispositivo de temporización llamado **generador de reloj**, el cual produce un tren periódico de **pulsos de reloj**. Los pulsos de reloj se distribuyen por todo el sistema de modo que los elementos de almacenamiento sólo se vean afectados al llegar cada pulso.

Los elementos de almacenamiento empleados en los circuitos secuenciales con reloj se llaman flip-flops. Un flip-flop es un dispositivo binario de almacenamiento que puede almacenar un bit de información. En la figura a continuación se ilustra el diagrama de bloques de un circuito secuencial sincrónico con reloj. Las salidas pueden provenir del circuito combinacional o de los flip-flops, o de ambos. El estado del flip-flop sólo puede cambiar durante una transición de pulso de reloj.



Circuito secuencial sincrónico con reloj

El comportamiento de un circuito secuencial **asincrónico** depende de las señales de entrada en cualquier instante dado y del orden en que cambian las entradas. Los elementos de almacenamiento que suelen usarse en los circuitos secuenciales asincrónicos son dispositivos de retardo de tiempo. En los sistemas asincrónicos tipo compuerta, los elementos de almacenamiento consisten en compuertas lógicas cuyo retardo de propagación hace posible el almacenamiento

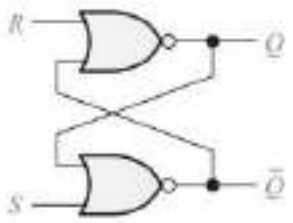
requerido. Así, un circuito secuencial asincrónico podría considerarse como un circuito combinacional con retroalimentación. Esta característica hace que puedan volverse inestables (Morris Mano, 2003).

Tema 4.2: Latches y flip-flops.

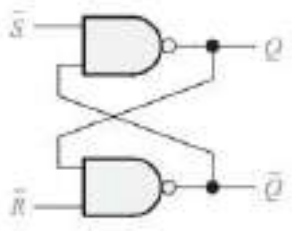
4.2.1 Latches

El **latch** (cerrojo) es un tipo de dispositivo de almacenamiento temporal de dos estados (**biestable** o **multivibrador**), que se suele agrupar en una categoría diferente a la de los flip-flops. Básicamente, los latches son similares a los flip-flops, ya que son también dispositivos de dos estados con capacidad de realimentación. La diferencia principal entre ambos tipos de dispositivos está en el método empleado para cambiar de estado.

Un **latch SR** (*Set-Reset*) con entrada activa a nivel ALTO se compone de dos puertas NOR acopladas, tal como se muestra en la figura a continuación a la izquierda; un latch con entrada activa a nivel BAJO está formado por dos puertas NAND conectadas tal como se muestra en la figura siguiente a la derecha. Observe que la salida de cada puerta se conecta a la entrada de la puerta opuesta. Esto origina la realimentación (feedback) regenerativa característica de todos los latches y flip-flops. (Floyd, 2006).



Latch SR con entrada activa a nivel ALTO



Latch $\bar{S}\bar{R}$ con entrada activa a nivel BAJO

El latch tiene dos estados útiles. Cuando las salidas *Q=1* y *Q'=0*, decimos que está en el **estado establecido** (*SET*). Cuando *Q=0* y *Q'=1*, está en el **estado restablecido** (*RESET*). Las salidas *Q* y *Q'* normalmente son una el complemento de la otra, pero si ambas entradas son 1 al mismo tiempo, se presenta un estado indefinido en el que ambas salidas son 0.

En condiciones normales, las dos entradas del latch permanecen en 0 a menos que se deba cambiar de estado. La aplicación momentánea de un 1 a la entrada *S* hace que el latch pase al estado establecido. La entrada *S* debe volver a 0 antes de cualquier otro cambio, para que no se presente el estado indefinido. Como se indica en la tabla de función de la siguiente figura, dos condiciones de entrada hacen que el circuito esté en el estado establecido. La primera condición (*S=1, R=0*) es la acción que debe efectuar la entrada *S* para poner el circuito en el estado establecido. Cuando la entrada activa se quita de *S*, el circuito permanece en el mismo estado. Una vez que las dos entradas regresan a 0, será posible cambiar

al estado restablecido aplicando momentáneamente un 1 a la entrada *R*. Luego puede quitarse el 1 de *R* y el circuito permanecerá en el estado restablecido. Así pues, cuando ambas entradas, *S* y *R*, son 0, el latch estará en el estado establecido o en el restablecido, dependiendo de cuál entrada fue 1 más recientemente (Morris Mano, 2003).

S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(Después de $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(Después de $S = 0, R = 1$)
1	1	0	0	

Tabla de función de Latch SR con compuertas NOR

Es posible modificar el funcionamiento del latch SR básico incluyendo una **entrada de control** adicional que determina cuándo puede cambiarse el estado del latch. En la figura siguiente se muestra un latch SR con entrada de control. Consiste en el latch SR básico y dos compuertas NAND adicionales. La entrada de control ***C*** actúa como señal de habilitación para las otras dos entradas.

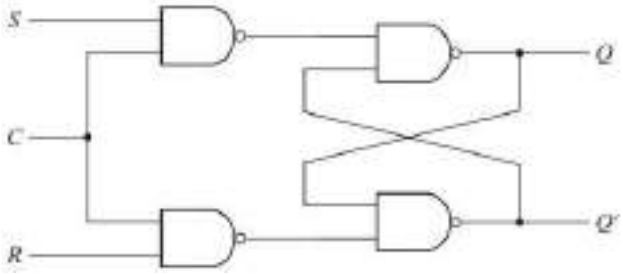


Diagrama de Latch SR con entrada de control

C	S	R	Siguiente estado de Q
0	X	X	Sin cambio
1	0	0	Sin cambio
1	0	1	$Q = 0$; estado restablecido
1	1	0	$Q = 1$; estado establecido
1	1	1	Indeterminado

Tabla de Latch SR con entrada de control

La salida de las compuertas NAND permanecerá en el nivel 1 lógico en tanto la entrada de control permanezca en 0. Ésta es la condición latente del latch SR. Cuando la entrada de control cambia a 1, se permite que la información de la entrada *S* o *R* afecte al latch SR. Se alcanza el estado establecido con *S=1, R=0* y *C=1*, y el estado restablecido con las entradas *S=0, R=1* y *C=1*. En ambos casos, cuando *C* regresa a 0, el circuito permanece en su estado actual (Morris Mano, 2003).

Una forma de eliminar la condición indeseable del estado indeterminado en el latch SR es garantizar que las entradas *S* y *R* nunca sean 1 al mismo tiempo. Esto se hace en el **latch D** que se ilustra en la siguiente figura. Este latch sólo tiene dos entradas: *D* (datos) y *C* (control).

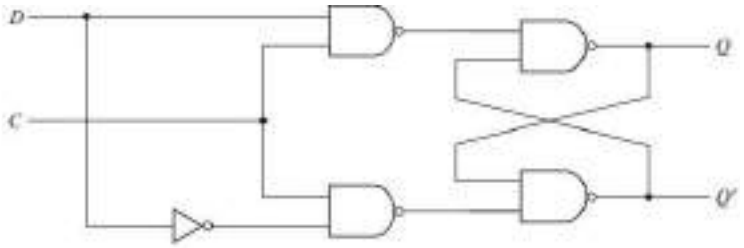


Diagrama lógico de Latch D

C	D	Siguiente estado de Q
0	X	Sin cambio
1	0	Q = 0: estado restablecido
1	1	Q = 1: estado establecido

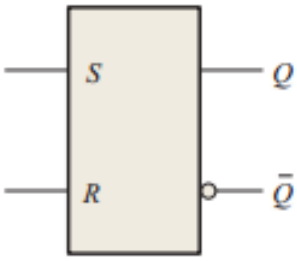
Tabla de función de Latch D

La entrada *D* pasa directamente a la entrada *S* y su complemento se aplica a la entrada *R*. En tanto la entrada de control esté en 0, el latch SR tendrá ambas entradas en el nivel 1 y el circuito no podrá cambiar de estado sea cual sea el valor de *D*. La entrada *D* se muestrea cuando *C=1*. Si *D=1*, la salida *Q* pasará a 1, colocando el circuito en el estado establecido. Si *D=0*, la salida *Q* pasará a 0, colocando el circuito en el estado restablecido.

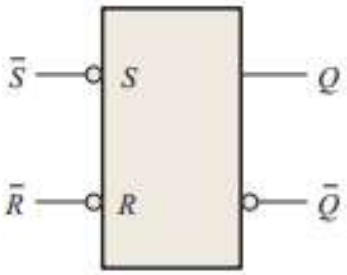
El latch D se llama así por su capacidad para almacenar datos en su interior. Es apropiado para usarse como almacenamiento temporal de información binaria entre una unidad y su entorno. La información binaria presente en la entrada de datos del latch D se transfiere a la salida *Q* cuando se habilita la entrada de control. También se lo conoce como **latch transparente** (Morris Mano, 2003).

Tal como lo hemos explicado, el latch es un dispositivo **asíncrono**, ya que no utiliza una señal de reloj para los cambios (Brown & Vranesic, 2006).

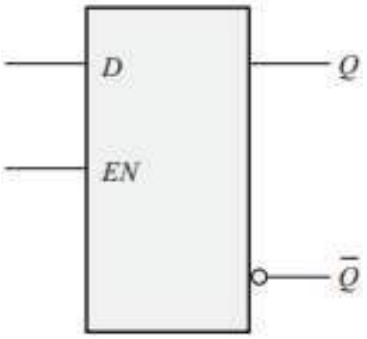
A continuación se ven los símbolos lógicos para los latchs explicados.



Símbolo lógico del latch SR



Símbolo lógico del latch $\bar{S}\bar{R}\bar{S}\bar{R}$



Símbolo lógico del latch D

4.2.2 Flip-flops

Los **flip-flops** son dispositivos **síncronos** de dos estados, también conocidos como **multivibradores biestables**. En este caso, el término síncrono significa que la salida cambia de estado únicamente en un instante específico de una entrada de disparo denominada **reloj** (CLK), la cual recibe el nombre de entrada de control.

Un **flip-flop disparado por flanco** cambia de estado con el flanco positivo (flanco de subida o **borde positivo**) o con el flanco negativo (flanco de bajada o **borde negaivo**) del impulso de reloj y es sensible a sus entradas sólo en esta transición del reloj (Floyd, 2006).

A continuación se representa la construcción de un **flip-flop D** con dos latches D y un inversor. El primer latch es el amo, y el segundo, el esclavo. El circuito muestrea la entrada *D* y cambia su salida *Q* únicamente en el borde negativo del reloj controlador (designado por *CLK* [*clock*]). Cuando el reloj es 0, la salida del inversor es 1. El latch esclavo queda habilitado y su salida *Q* es igual a la salida *Y* del amo. El latch amo queda inhabilitado porque *CLK=0*.

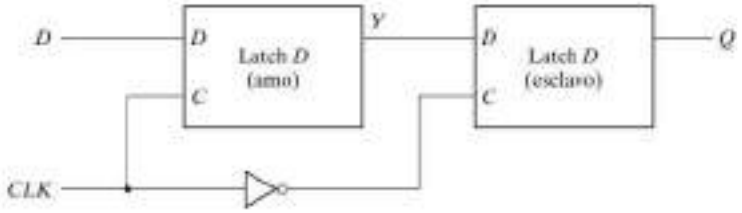
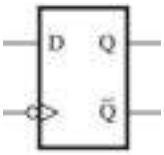


Diagrama del circuito Flip-flop D amo-esclavo



Símbolo gráfico del Flip-flop D

Cuando el pulso de entrada cambia al nivel de 1 lógico, el dato de la entrada *D* externa se transfiere al amo. El esclavo, empero, estará inhabilitado en tanto el reloj permanezca en el nivel 1, porque su entrada *C* es 0. Cualquier cambio en la entrada hará que cambie la salida *Y* del amo, pero no afectará la salida del esclavo. Cuando el pulso vuelva a 0, el amo quedará inhabilitado y aislado de la entrada *D*. Al mismo tiempo, el esclavo estará habilitado y el valor *Y* se transferirá a la salida *Q* del flip-flop. Así, la salida del flip-flop sólo puede cambiar durante la transición del reloj de 1 a 0, es decir, la salida sólo puede cambiar durante el **borde negativo** del reloj.

También es posible diseñar el circuito de modo que la salida del flip-flop cambie en el **borde positivo** del reloj. Esto sucede en un flip-flop que tiene un inversor adicional entre la terminal *CLK* y la unión entre el otro inversor y la entrada *C* del latch amo. Un flip-flop semejante se dispara con un pulso negativo, de modo que el borde negativo del reloj afecta al amo, y el positivo, al esclavo y a la terminal de salida. (Floyd, 2006).

Hay tres operaciones que pueden efectuarse con un flip-flop: establecerlo en 1, restablecerlo a 0 y complementar su salida. El **flip-flop JK** realiza las tres operaciones. El diagrama de circuito de un flip-flop JK construido con un flip-flop D y compuertas se reproduce en la figura a continuación. El símbolo gráfico del flip-flop JK es similar al del flip-flop D, excepto que ahora las entradas están marcadas con Jy K.

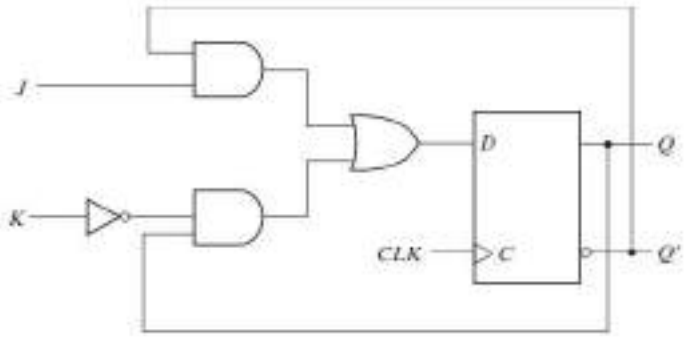
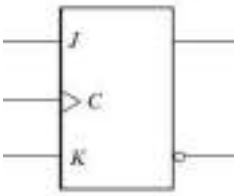


Diagrama del circuito Flip-flop JK

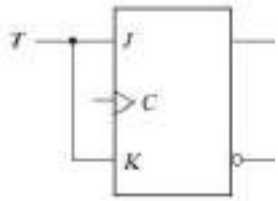


Símbolo gráfico del Flip-flop JK

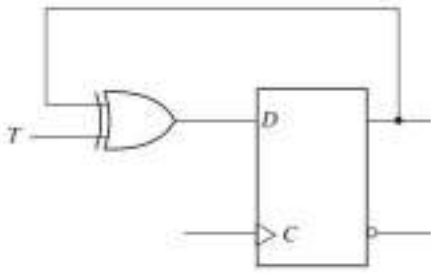
La entrada *J* establece el flip-flop en 1, la entrada *K* lo restablece a 0 y, cuando ambas entradas están habilitadas, la salida se complementa. Esto se verifica investigando el circuito aplicado a la entrada *D*: *D=JQ'+K'Q*.

- Cuando *J=1* y *K=0, D=Q'+Q=1*, así que el siguiente borde del reloj establece la salida en 1.
- Cuando *J=0* y *K=1, D=0*, así que el siguiente borde del reloj restablece la salida a 0.
- Cuando *J=1* y *K=1, D=Q'*, y el siguiente borde del reloj complementa la salida. Cuando *J=K=0, D=Q*, y el borde de reloj no altera la salida.

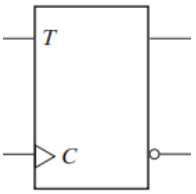
El **flip-flop T** (*toggle*) es un flip-flop **complementador** y se puede implementar con un flip-flop JK si se conectan entre sí las entradas Jy K, como se observa en la a continuación en la figura de la izquierda. Cuando *T=0* (*J=K=0*), un borde de reloj no modifica la salida. Cuando *T=1* (*J=K=1*), un borde de reloj complementa la salida. El flip-flop T es útil para diseñar contadores binarios.



Flip-flop T con un flip-flop JK



Flip-flop T con un flip-flop D



Símbolo gráfico del Flip-flop T

El flip-flop Tse puede construir con un flip-flop D y una compuerta OR exclusivo, como se indica en la figura del centro. La expresión para la entrada *D* es *D=T⊕Q=TQ'+T'Q*. Cuando *T=0*, entonces *D=Q*, y la salida no cambia. Cuando *T=1*, entonces *D=Q'* y la salida se complementa. El símbolo gráfico de este flip-flop tiene una ***T*** en la entrada.

Una tabla característica define las propiedades lógicas de un flip-flop describiendo su funcionamiento en forma tabular. A continuación se muestran las tablas características o de función de los flip-flips que hemos visto (Morris Mano, 2003).

Flip-Flop *D*

D	$Q(t+1)$	
0	0	Restablecer
1	1	Establecer

Flip-Flop *JK*

J	K	$Q(t+1)$	
0	0	$Q(t)$	Sin cambio
0	1	0	Restablecer
1	0	1	Establecer
1	1	$Q'(t)$	Complementar

Flip-Flop *T*

T	$Q(t+1)$	
0	$Q(t)$	Sin cambio
1	$Q'(t)$	Complementar

Tablas de función de los flip-flops

Tema 4.3: Descripción de los flip-flops con el lenguaje VHDL.

Una vez visto cómo funcionan los latch y flip-flops, vamos a ver como se los puede codificar con VHDL.

4.3.1 Descripción de latches

A continuación se define una entidad llamada ***latch***, la cual tiene las entradas *D* y *Clk* y la salida *Q*. El proceso utiliza una instrucción *if-then-else* para definir el valor de la salida *Q*. Cuando *Clk=1*, *Q* toma el valor de *D*. Para el caso en que *Clk* no es 1, el código no especifica qué valor debe tener *Q*; por consiguiente, *Q* conservará su valor actual en este caso, y el código describe un ***latch D* asíncrono**.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY latch IS
     PORT ( D, Clk : IN STD LOGIC ;
          Q : OUT STD LOGIC) ;
END latch ;

ARCHITECTURE Behavior OF latch IS
BEGIN
     PROCESS ( D, Clk )
     BEGIN
          IF Clk='1' THEN
               Q<=D;
          END IF ;
     END PROCESS ;
END Behavior ;
```

La lista de sensibilidad del proceso incluye tanto *Clk* como *D* porque estas señales pueden causar un cambio en el valor de la salida *Q* (Brown & Vranesic, 2006).

4.3.2 Descripción de flip-flops

El siguiente código define una entidad llamada ***flipflop***, que es un ***flip-flop D*** disparado por flanco positivo. El código es idéntico al del *latch* antes mostrado, con dos excepciones. Primera, la lista de sensibilidad del proceso contiene sólo la señal de relojque puede causar un cambio en la salida *Q*. Segunda, la instrucción *if-then-else* utiliza una condición diferente de la empleada en el latch. La sintaxis *Clock'EVENT* usa un constructor de VHDL llamado *atributo*. Un atributo se refiere a la propiedad de un objeto, como una señal. En este caso el atributo *'EVENT* se refiere a cualquier c ambio en la señal *Clock*. Combinar la condición Clock'EVENT con la condición Clock1 significa que "el valor de la señal *Clock* acaba de cambiar, y el valor ahora es igual a 1". Por consiguiente, la condición se refiere a un flanco positivo del reloj. Por lo tanto, el código describe un *flip-flop D* disparado por flanco positivo.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY flipflop IS
      PORT ( D, Clock : IN STD LOGIC ;
            Q : OUT STD LOGIC) ;
END flipflop

ARCHITECTURE Behavior OF flipflop IS
BEGIN
      PROCESS ( Clock )
      BEGIN
            IF Clock'EVENT AND Clock='1' THEN
                  Q<=D;
            END IF ;
      END PROCESS ;
END flipflop ;
```

El siguiente es un código opcional para un *flip-flop D*. En este caso se emplea una sintaxis distinta de la anterior. Utiliza la instrucción *WAIT UNTIL Clock'EVENT AND Clock='1'*, que produce el mismo efecto que la instrucción *IF* del código anterior. Un proceso que usa una instrucción *WAIT UNTIL* es un caso especial porque se omite la lista de sensibilidad (Brown & Vranesic, 2006).

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY flipflop IS
      PORT ( D, Clock : IN STD LOGIC ;
            Q : OUT STD LOGIC) ;
END flipflop

ARCHITECTURE Behavior OF flipflop IS
```

```
BEGIN
      PROCESS
      BEGIN
            WAIT UNTIL Clock'EVENT AND Clock'1' ;
                  Q<=D;
            END IF ;
      END PROCESS ;
END flipflop ;
```

Sabiendo cuales son las entradas y salidas que se deben declarar en una entidad, se pueden implementar otros circuitos. Por ejemplo, el proceso de un ***flip-flop JK*** podría ser el siguiente (Machado & Borromero, 2010):

```
P_JK1: PROCESS (Clk)
BEGIN
      IF Clk'EVENT and Clk='1'THEN
            IF J='1' AND K='1' THEN
                  Q <= NOT Q ;
            ELSIF J='1' AND K='0' THEN
                  Q <= '1' ;
            ELSIF J='0' AND K='1' THEN
                  Q <= '0' ;
            ELSE
                  Q <= Q ;
            END IF ;
      END IF ;
END PROCESS ;
```

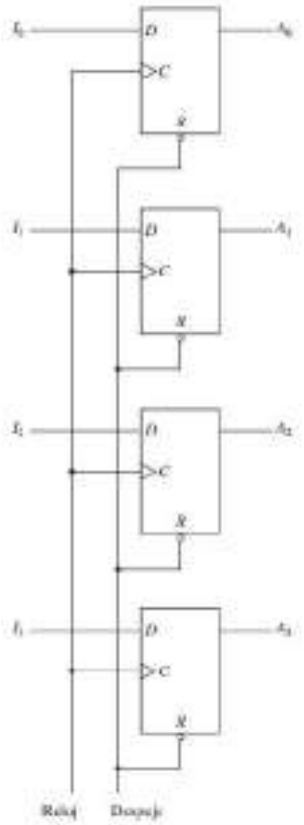
Tema 4.4: Registros y contadores binarios sincrónicos y asincrónicos.

4.4.1 Registros

Cuando un conjunto de *n* flip-flops se usa para almacenar *n* bits de información, como un número de *n* bits, nos referimos a él como un **registro**. Para cada flip-flop de un registro se usa un **reloj común**, y cada flip-flop funciona como se describió en el tema anterior. El término *registro* es una convención para referirse a las estructuras de *n* bits que se componen de flip-flops (Brown & Vranesic, 2006).

Además de los flip-flops, un registro puede tener compuertas combinacionales que realizan ciertas tareas de procesamiento de datos. En su definición más amplia, un **registro** consiste en un grupo de flip-flops y compuertas que efectúan su transición. Los flipflops contienen la información binaria y las compuertas determinan cómo esa información se transfiere al registro.

Hay diversos tipos de registros. El más sencillo consiste únicamente en flipflops, sin compuertas. La siguiente figura muestra uno registro construido con cuatro flip-flops D.

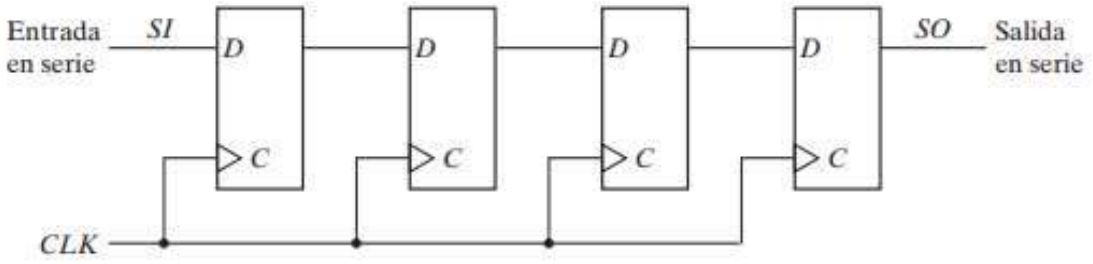


Registro de cuatro bits

La entrada de reloj, común a todos los flip-flops, los dispara en el flanco positivo de cada pulso, y los datos binarios disponibles en las cuatro entradas se transfieren en el registro de cuatro bits. Es posible muestrear las cuatro salidas en cualquier momento para obtener la información binaria almacenada en el registro. La entrada de **despeje** (*clear*) se conecta a la entrada *R* (*reset*) de los cuatro flip-flops. Cuando esta entrada cambia a 0, todos los flip-flops se restablecen asincrónicamente. La entrada *clear* sirve para poner en ceros el registro antes de que comience a funcionar con reloj. Las entradas *R* se deben mantener en 1 lógico durante el funcionamiento normal con reloj. Se utiliza tanto *clear* como *reset* para indicar la transferencia del registro al estado de ceros.

Un **registro de desplazamiento** es un registro capaz de desplazar su información binaria en una dirección o en la otra. La configuración lógica de un registro de desplazamiento consiste en una cadena de flip-flops en cascada, con la salida de un flip-flop conectada a la entrada del siguiente flip-flop. Todos los flip-flops reciben pulsos de reloj comunes, que activan el desplazamiento de una etapa a la siguiente.

El registro de desplazamiento más sencillo posible usa sólo flip-flops, como se indica en la figura siguiente. La salida de un flip-flop dado se conecta a la entrada *D* del flip-flop que está a su derecha. Cada pulso de reloj desplaza el contenido del registro una posición de bit a la derecha. La entrada en serie determina qué sucede en el flip-flop de la extrema izquierda durante el desplazamiento. La salida en serie se toma de la salida del flip-flop de la extrema derecha (Morris Mano, 2003).



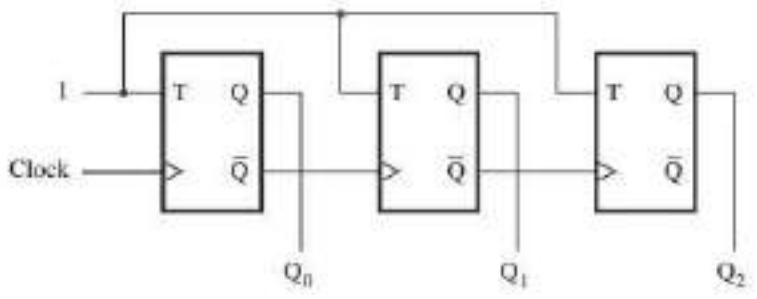
Registro de desplazamiento de cuatro bits

4.4.2 Contadores

Un **contador** es básicamente un registro que pasa por una sucesión predeterminada de estados. Las compuertas del contador están conectadas de tal manera que producen la sucesión prescrita de estados binarios. Aunque los contadores son un tipo especial de registros, es común distinguirlos dándoles otro nombre.

La idea principal del contador es que puedan aumentar o disminuir un conteo de 1 en 1. Se pueden construir contadores asíncronos o síncronos.

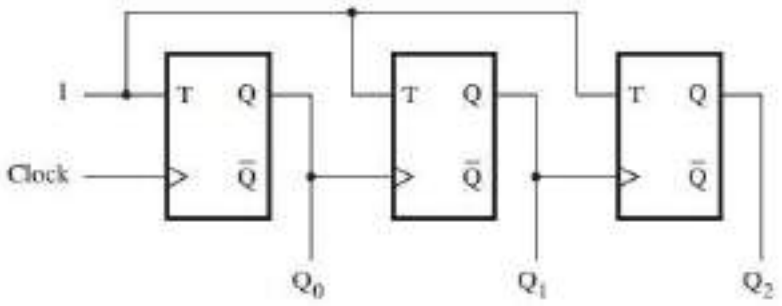
Un **contador asíncrono** es aquél en el que los flip-flops del contador no cambian de estado exactamente al mismo tiempo, dado que no comparten el mismo impulso de reloj. En la figura siguiente se describe un **contador asíncrono ascendente** de tres bits, construido con flip-flops *T*, que puede contar de 0 a 7. Las entradas del reloj de los tres flip-flops están conectadas en cascada. La entrada *T* de cada flip-flop está conectada a una constante 1, lo que significa que el estado del flip-flop se invertirá (cambiará a un segundo estado) en cada flanco positivo de su reloj.



Circuito de un contador asíncrono ascendente de 3 bits

Estamos suponiendo que el propósito de este circuito es contar el número de pulsos que ocurren en la entrada principal llamada *Clock*. Por tanto, la entrada del reloj del primer flip-flop está conectada a la línea *Clock*. Las entradas del reloj de los otros dos flip-flops están manejadas por la salida *Q*' del flip-flop anterior. Por consiguiente, alternan su estado siempre que el fl ip-flop precedente cambia su estado de *Q=1* a *Q=0*, lo que resulta en el flanco positivo de la señal Q. El contador mostrado tiene tres etapas, y cada una consta de un solo flip-flop. Sólo la primera etapa responde directamente a la señal *Clock*, mientras que las otras dos responden después de un retraso adicional. La secuencia de conteo de este circuito es 0, 1, 2, 3, 4, 5, 6, 7, 0, 1 y así sucesivamente.

En la figura a continuación se muestra un **contador asíncrono descendente**. La única diferencia con el anterior es que en esta última 7las entradas del reloj del segundo y tercer flip-flops están manejadas por las salidas Q de las etapas precedentes, en vez de estarlo por las salidas Q'. Este circuito cuenta en la secuencia 0, 7, 6, 5, 4, 3, 2, 1, 0, 7 y así sucesivamente (Brown & Vranesic, 2006).



Un contador asíncrono descendente de 3 bits

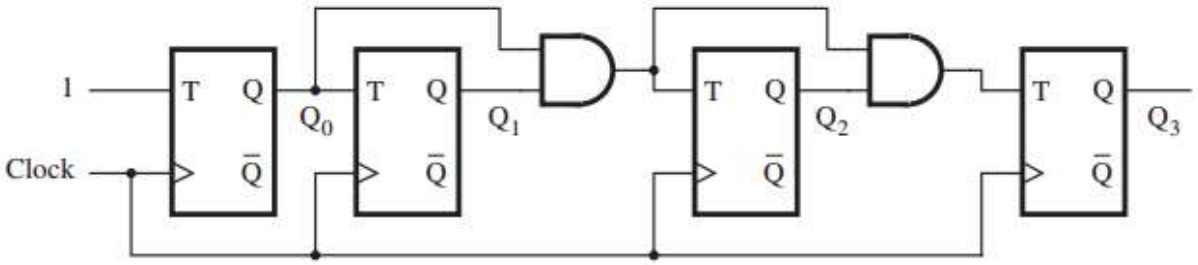
Un **contador síncrono** es aquel en que se aplican pulsos de reloj a las entradas de todos los flip-flops. Un mismo reloj dispara todos los flip-flops simultáneamente en vez de hacerlo uno por uno. La decisión de si un flip-flop debe complementarse o no depende de los valores de las entradas de datos (Morris Mano, 2003).

En la tabla a continuación se muestra el contenido de un **contador síncrono ascendente** de tres bits para ocho ciclos del reloj consecutivos, suponiendo que el conteo empieza en 0. Al observar el patrón de bits de cada fila de la tabla es claro que el bit Q_0 cambia en cada ciclo del reloj. El bit Q_1 cambia sólo cuando Q_0=1. El bit Q_2 cambia únicamente cuando Q_1 y Q_0 son iguales a 1. En general, para un contador ascendente de ***n*** bits, un flip-flop dado cambia su estado sólo cuando todos los flip-flops anteriores están en el estado *Q*=1. Por consiguiente, si usamos flip-flops T para hacer el contador, entonces las entradas T se definen como $\boldsymbol{T_n=Q_0Q_1\ldots Q_{n-1}}$.

Ciclo del reloj	Q_2	Q_1	Q_0	
0	0	0	0	
1	0	0	1	
2	0	1	0	Q_1 cambia
3	0	1	1	
4	1	0	0	Q_2 cambia
5	1	0	1	
6	1	1	0	
7	1	1	1	
8	0	0	0	

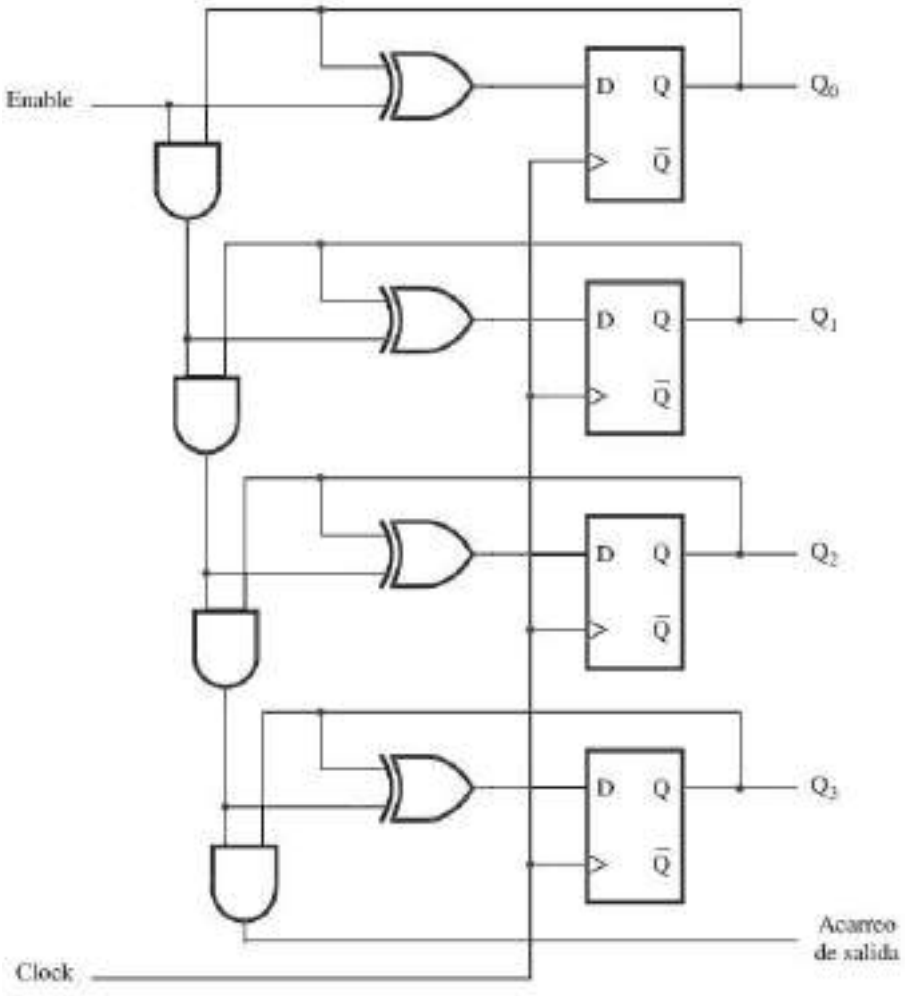
Tabla de funcionamiento del contador síncrono ascendente

Un ejemplo de un contador de cuatro bits basado en estas expresiones aparece en la siguiente figura. En vez de usar compuertas AND de tamaño aumentado para cada etapa, lo que puede suscitar problemas, empleamos un arreglo factorizado, que no reduce la respuesta del contador. Un cambio en el valor de Q_0 debe propagarse por varias compuertas AND para llegar a los flip-flops en las etapas superiores del contador, lo cual requiere cierta cantidad de tiempo, que no debe exceder el periodo del reloj.



Un contador síncrono ascendente de cuatro bits

Es posible también construir contadores usando otros tipos de flip-flops. Los flip-flops JK pueden emplearse exactamente igual que los flip-flops T, ya que si las entradas *J* y *K* se unen, un flip-flop JK se convierte en un flip-flop T. La figura siguiente muestra una posible implementación de un contador de 4 bits con flip-flops D. Si suponemos que *Enable*=1, entonces las entradas *D* de los flip-flops se definen mediante las expresión $D_i=Q_i \oplus Q_{i-1}Q_{i-2}\ldots Q_1Q_0$, o más precisamente, $D_i=Q_i \oplus Q_{i-1}.Q_{i-2}.\ldots Q_1.Q_0.Enable$.



Un contador de cuatro bits con flip-flops D

El funcionamiento del contador se basa en el principio de que el estado del flip-flop en una etapa ***i*** cambia sólo si todos los flip-flops anteriores están en el estado *Q*=1 (Brown & Vranesic, 2006).

Tema 4.5: Descripción de registros y contadores con leguaje de descripción del Hardware.

En esta sección veremos la descripción de algunos de los circuitos de registros y contadores que hemos visto.

4.5.1 Descripción de registros

Una forma para hacer un **registro** sería mediante un código que implemente jerárquicamente ***n*** instancias de flip-flops como los que vimos en temas anteriores. Un enfoque más simple se muestra a continuación. Utiliza un código similar, excepto que la entrada *D* y la salida *Q* se definen como señales multibit. El código representa un **registro de ocho bits** con borrado asíncrono

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;

ENTITY reg8 IS
      PORT ( D : IN STD LOGIC VECTOR(7 DOWNTO 0) ;
            Resetn, Clock : IN STD LOGIC ;
            Q : OUT STD LOGIC VECTOR(7 DOWNTO 0) ) ;
END reg8 ;

ARCHITECTURE Behavior OF reg8 IS
BEGIN
      PROCESS ( Resetn, Clock )
      BEGIN
            IF Resetn='0' THEN
                  Q<="00000000" ;
            ELSIF Clock'EVENT AND Clock='1' THEN
                  Q<=D;
            END IF ;
      END PROCESS ;
END Behavior ;
```

El código del **registro de *n* bits** es mostrado a continuación. Aquí se extiende el código anterior para incluir un parámetro que establezca el número de flip-flops. El parámetro es un entero *N* que se define usando el constructor de VHDL llamado *GENERIC*. El valor de *N* se establece en 16 usando el operador de asignación **:=**.Al cambiar este parámetro, el código puede representar un registro de cualquier tamaño. Si el registro se declara como un componente, entonces sirve como subcircuito en otro código.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;

ENTITY regn IS
      GENERIC(N:INTEGER :16 ) ;
      PORT ( D : IN STD LOGIC VECTOR(N-1 DOWNTO 0);
            Resetn, Clock : IN STD LOGIC ;
            Q : OUT STD LOGIC VECTOR(N-1 DOWNTO 0));
END regn;

ARCHITECTURE Behavior OF regn IS
BEGIN
      PROCESS ( Resetn, Clock )
      BEGIN
            IF Resetn='0' THEN
                  Q<=(OTHERS=>'0') ;
            ELSIF Clock'EVENT AND Clock='1' THEN
                  Q<=D;
            END IF ;
```

```
      END PROCESS ;
END Behavior ;
```

La instrucción que inicializa todos los bits de *Q* en 0 usa la extraña sintaxis *Q<=(OTHERS=<'0')*. La sintaxis *(OTHERS=>'0')* tiene como resultado un dígito '0' que se asigna a cada uno de los bits de *Q*, independientemente de cuántos bits tenga *Q*.

El código de VHDL que represente un **registro de desplazamiento de cuatro bits** se muestra a continuación. Consiste en escribir código jerárquico que use cuatro subcircuitos. Cada subcircuito se compone de un *flip-flop D* con un *multiplexor dos a uno* conectado a la entrada *D*. En el código se define la entidad llamada *muxdff*, que representa este subcircuito. Los dos datos de enrada se llaman D_0 y D_1,y se seleccionan por medio de la entrada *Sel*.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all ;

ENTITY shift4 IS
      PORT ( R : IN STD LOGIC VECTOR(3 DOWNTO 0) ;
            L, w, Clock : IN STD LOGIC ;
            Q : BUFFER STD LOGIC VECTOR(3 DOWNTO 0) ) ;
END shift4;

ARCHITECTURE Structure OF shift4 IS
      COMPONENT muxdff
            PORT ( D0, D1, Sel, Clock : IN STD LOGIC ;
                  Q : OUT STD LOGIC ) ;
      END COMPONENT ;
BEGIN
      Stage3: muxdff PORT MAP ( w, R(3), L, Clock, Q(3) ) ;
      Stage2: muxdff PORT MAP ( Q(3), R(2), L, Clock, Q(2) ) ;
      Stage1: muxdff PORT MAP ( Q(2), R(1), L, Clock, Q(1) ) ;
      Stage0: muxdff PORT MAP ( Q(1), R(0), L, Clock, Q(0) ) ;
END Structure;
```

4.5.2 Descripción contadores

A continuación se muestra el código para un **contador ascendente de cuatro bits** que tiene una entrada de inicialización, *Resetn*, y una entrada de habilitación *E*. En el cuerpo de arquitectura los flip-flops del contador se representan mediante la señal llamada *Count*. La instrucción del proceso (*process*) especifica un *reset* asíncrono de Count cuando *Resetn*=0. La cláusula ELSIF especifica que en el flanco positivo del reloj, si *E*=1, el conteo se incrementa. Si *E*=0, el código asigna explícitamente *Count*<=*Count*. Las salidas *Q* se asignan al valor de *Count* al final del código.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std logic unsigned.all ;

ENTITY upcount IS
     PORT ( Clock, Resetn, E : IN STD LOGIC ;
          Q : OUT STD LOGIC VECTOR (3 DOWNTO 0)) ;
END upcount ;

ARCHITECTURE Behavior OF upcount IS
     SIGNAL Count : STD LOGIC VECTOR (3 DOWNTO 0) ;
BEGIN
     PROCESS ( Clock, Resetn )
     BEGIN
          IF Resetn='0' THEN
               Count<="0000" ;
          ELSIF (Clock'EVENT AND Clock='1') THEN
               IF E='1' THEN
                    Count<=Count+1;
               ELSE
                    Count<=Count ;
               END IF ;
          END IF ;
     END PROCESS ;
     Q<=Count
END Behavior ;
```

Ahora se muestra el código para un **contador descendente** llamado *downcnt*. Para facilitar el cambio del conteo inicial, se define como un parámetro GENERIC llamado *modulus*. En el flanco positivo del reloj, si *L*=1 el contador se carga con los valores *modulus*–1, y si L=0, el conteo disminuye. El contador también incluye una entrada de habilitación, *E*. Establecer *E*=1permite que el conteo disminuya cuando ocurre un flanco activo del reloj.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all;

ENTITY downcnt IS
     GENERIC ( modulus : INTEGER :=8);
     PORT ( Clock, L, E : IN STD LOGIC ;
          Q : OUT INTEGER RANGE 0 TO modulus-1) ;
END upcount ;

ARCHITECTURE Behavior OF downcnt IS
     SIGNAL Count : INTEGER RANGE 0 TO modulus-1;
BEGIN
     PROCESS
     BEGIN
```

```
            WAIT UNTIL (Clock'EVENT AND Clock='1') ;
            IF L='1' THEN
                  Count<=modulus-1;
            ELSE
                  IF E'1' THEN
                        Count<=Count-1;
                  END IF ;
            END IF ;
      END PROCESS ;
      Q<=Count
END Behavior ;
```

PROBLEMAS DE SISTEMAS ELECTRÓNICOS

INVERNADERO AUTOMATIZADO CON CONTROL DE TEMPERATURA Y HUMEDAD: SIMULANDO LA APERTURA DE VENTANAS O ENCENDIDO DE VENTILADORES

Un invernadero automatizado con control de temperatura y humedad permite optimizar las condiciones de crecimiento para las plantas de manera eficiente. A través de sensores, se monitorean constantemente los niveles de temperatura y humedad dentro del invernadero, garantizando que se mantengan dentro de los rangos ideales para el desarrollo de las especies cultivadas.

Este sistema automatizado puede simular la apertura de ventanas o el encendido de ventiladores cuando se detectan niveles fuera de los límites establecidos. Si la temperatura es demasiado alta, el sistema abre automáticamente las ventanas o activa los ventiladores para bajar la temperatura. De igual manera, si la humedad cae por debajo de lo deseado, se pueden activar mecanismos que mantengan un ambiente óptimo.

SABÍAS QUE

Un invernadero automatizado puede ahorrar hasta un **30% de energía** al controlar automáticamente la apertura de ventanas y ventiladores, manteniendo el ambiente perfecto para las plantas.

La automatización en invernaderos no solo facilita el proceso de cultivo, sino que también mejora la eficiencia en el uso de recursos como agua y energía. Además, al contar con un sistema de monitoreo constante y ajustes automáticos, los agricultores pueden asegurarse de que las plantas siempre estén en condiciones óptimas sin la necesidad de intervención constante.

DISEÑO DE PROTOTIPO

El circuito de este proyecto automatizado se basa en la combinación de varios componentes para monitorear la temperatura y la humedad, así como para accionar mecanismos de control, como el ventilador y las ventanas. A continuación, se describe el diseño del circuito y la función de cada componente:

1. **Sensor DHT22**: Este sensor está conectado al pin digital 2 y es responsable de medir tanto la temperatura como la humedad del entorno. El sensor tiene tres pines principales: el de alimentación (VCC), el de tierra (GND) y el de datos (DATA), que se conecta al microcontrolador (en este caso, el pin 2). Proporciona las lecturas necesarias para que el sistema tome decisiones sobre cuándo activar los otros componentes.

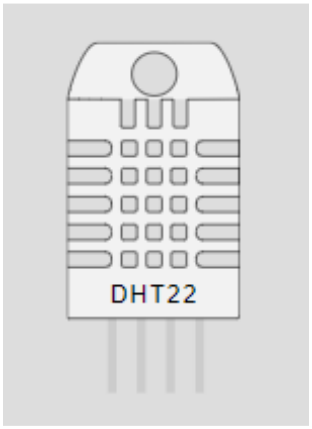


Figura 1: Sensor DHT22

2. **LED que simula el ventilador**: El ventilador está representado por un LED conectado al pin digital 3. El LED enciende o apaga según la temperatura registrada por el sensor DHT22. El circuito del LED incluye una resistencia (típicamente de 220 ohmios) para limitar la corriente y evitar que el LED se queme.

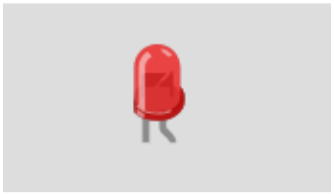


Figura 2: LED

3. **Servomotores para ventanas**: Hay dos servomotores conectados a los pines 9 y 10 que controlan la apertura y el cierre de las ventanas. Los servos se alimentan desde la misma fuente de alimentación que el microcontrolador, y los pines de control (9 y 10) permiten que el microcontrolador envíe señales PWM para mover las ventanas a la posición deseada según las lecturas de temperatura y humedad.

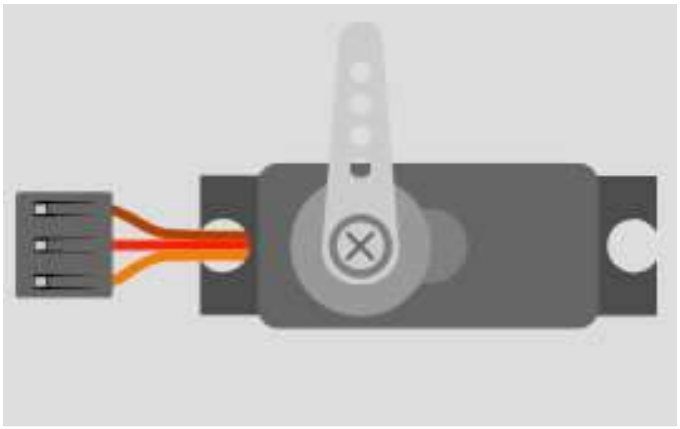


Figura 3: Servomotores

Desarrollo del código:

En la unidad anterior, aprendimos que es posible utilizar diferentes simuladores para nuestros proyectos de electrónica. Ahora, veamos de forma general cómo debe estar estructurado nuestro código, el cual podrá ser adaptado fácilmente en el simulador de tu preferencia.

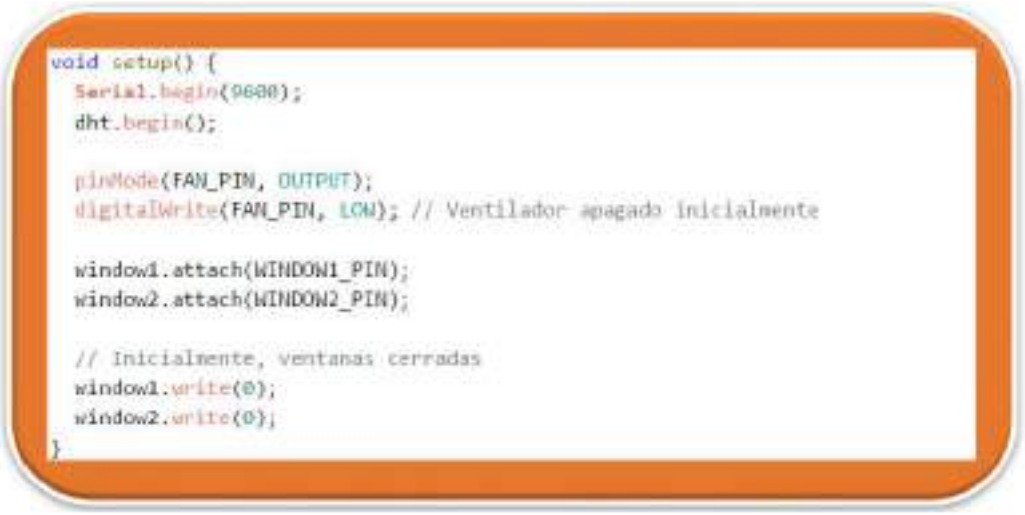
Estructura básica del código:

1. **Declaración de librerías y variables**:
    - Se incluyen las librerías DHT.h para el sensor de temperatura y humedad DHT22 y Servo.h para controlar los servomotores de las ventanas.
    - Se definen los pines del sensor DHT22, del ventilador (simulado con un LED), y de los servomotores que controlan las ventanas.
    - Se crean objetos para el sensor DHT22 y los servomotores.

```
#include <DHT.h>
#include <Servo.h>

#define DHTPIN 2     // Pin de datos del sensor DHT22
#define DHTTYPE DHT22   // Tipo de sensor DHT22
#define FAN_PIN 3    // Pin del LED que simula el ventilador
#define WINDOW1_PIN 9 // Pin del servomotor para la ventana 1
#define WINDOW2_PIN 10 // Pin del servomotor para la ventana 2

DHT dht(DHTPIN, DHTTYPE);

Servo window1;
Servo window2;
```

Configuración de pines y componentes (setup):

- En la función setup(), se inicializan la comunicación serial y el sensor DHT22.
- Se configura el pin del ventilador como salida y se apaga inicialmente.
- Se conectan los servomotores a sus respectivos pines y se colocan inicialmente en la posición de “ventanas cerradas” (0 grados).



```
void setup() {
  Serial.begin(9600);
  dht.begin();

  pinMode(FAN_PIN, OUTPUT);
  digitalWrite(FAN_PIN, LOW); // Ventilador apagado inicialmente

  window1.attach(WINDOW1_PIN);
  window2.attach(WINDOW2_PIN);

  // Inicialmente, ventanas cerradas
  window1.write(0);
  window2.write(0);
}
```

Ciclo principal (loop):

- En la función loop(), el sistema espera 2 segundos entre cada lectura para dar tiempo a que el sensor DHT22 obtenga las mediciones de temperatura y humedad.
- Se realizan las lecturas de temperatura y humedad. Si las lecturas fallan, se muestra un error en el monitor serial.
- Las lecturas exitosas se imprimen en el monitor serial, mostrando la humedad en porcentaje y la temperatura en grados Celsius.
- Basado en la temperatura:
    - Si la temperatura es mayor a 30°C, se enciende el ventilador (LED).
    - Si es menor o igual a 30°C, el ventilador se apaga.

- Basado en la temperatura y humedad:
    - Si la temperatura es mayor a 25°C o la humedad es mayor a 60%, se abren ambas ventanas (moviendo los servomotores a 90 grados).
    - Si la temperatura es menor o igual a 25°C y la humedad es menor o igual a 60%, las ventanas se cierran (0 grados).

```
void loop() {
  delay(2000); // Esperar entre lecturas

  // Leer temperatura y humedad
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  // Verificar si la lectura falló
  if (isnan(h) || isnan(t)) {
    Serial.println("Error al leer el sensor DHT22.");
    return;
  }

  // Mostrar en el monitor serial
  Serial.print("Humedad: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperatura: ");
  Serial.print(t);
  Serial.println(" *C");

  // Controlar ventilador (LED)
  if (t > 30) {
    digitalWrite(FAN_PIN, HIGH); // Encender ventilador
  } else {
    digitalWrite(FAN_PIN, LOW);  // Apagar ventilador
  }
```

```
// Controlar ventanas (servomotores)
 if (t > 25) {
   window1.write(90); // Abrir ventana 1
   window2.write(90); // Abrir ventana 2
 } else {
   window1.write(0);  // Cerrar ventana 1
   window2.write(0);  // Cerrar ventana 2
 }

 if (h > 80) {
   // Si la humedad es alta, abrir las ventanas
   window1.write(90);
   window2.write(90);
 } else {
   // Si la humedad es baja, mantener las ventanas cerradas
   window1.write(0);
   window2.write(0);
 }
}
```

Con esta estructura básica en mente, podrás implementar cualquier proyecto de electrónica, y adaptar tu código según el simulador que estés utilizando, ya sea Tinkercad, Proteus, o Wokwi.

Conexión de componentes:

Los componentes y pines siempre tendrán la misma estructura tanto en los simuladores como en la versión física. Aquí te proporcionamos una guía sobre cómo deben conectarse. Sigue los mismos pasos si vas a utilizar el código proporcionado, y si cambias algún pin, no olvides hacer el ajuste correspondiente en el código.

Componentes	Abreviatura
Servo motor	Servo1, Servo2
Led	Led1
Sensor DHT22	Dht1

Servo1: GND conectado a uno: GND
Servo1: V+ conectado a uno :5V
Servo1: PWM conectado a uno :9
Servo2: GND conectado a uno: GND
Servo2: V+ conectado a uno:5V
Servo2: PWM conectado a uno:10
Dht1: VCC conectado a uno:5V
Dht1: SDA conectado a uno:2
Dht1: GND conectado a uno: GND
Led1:C conectado a uno: GND
Led1: A conectado a uno:3

Diagrama esquemático:

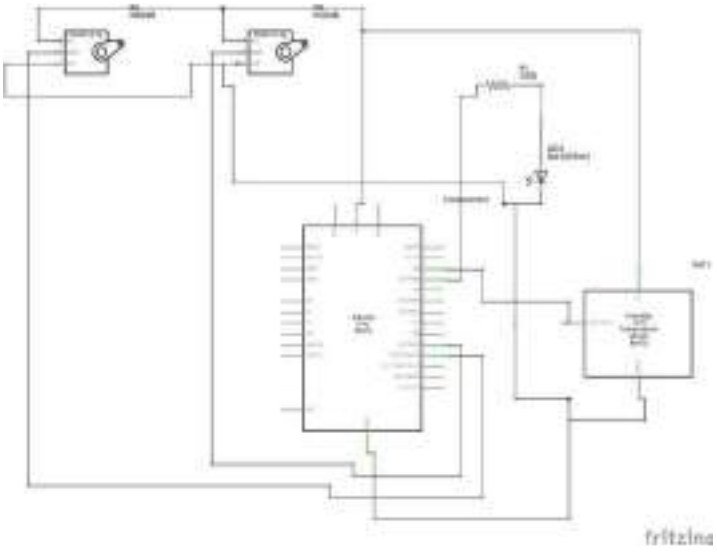


Figura 4: diagrama esquemático del circuito proporcionado por el software fritzing

PROBLEMAS DE SISTEMAS ELECTRÓNICOS

MEDIDOR DE ENERGÍA PARA HOGAR INTELIGENTE: MONITOREANDO EL CONSUMO DE ENERGÍA EN TIEMPO REAL CON ALERTAS

El Medidor de Energía para Hogar Inteligente es un dispositivo diseñado para monitorear en tiempo real el consumo eléctrico en el hogar, brindando a los usuarios información precisa sobre su uso energético. Este sistema permite identificar patrones de consumo y detectar posibles anomalías, contribuyendo a la optimización de los recursos y el ahorro de energía.

El medidor inteligente no solo registra el consumo, sino que también puede emitir alertas cuando se superan ciertos umbrales establecidos por el usuario, ayudando a evitar sobrecargas o facturas inesperadas. Además, se integra fácilmente con otras tecnologías del hogar inteligente, permitiendo un control eficiente del uso energético desde cualquier lugar.

Gracias a su capacidad de análisis en tiempo real, este medidor se convierte en una herramienta clave para aquellos que buscan un hogar más eficiente y sostenible, proporcionando información accesible para una mejor toma de decisiones.

DISEÑO DE PROTOTIPO

El diseño del circuito para este proyecto de **Medidor de Energía** utiliza varios componentes clave que simulan el monitoreo del consumo de energía y emiten alertas visuales y sonoras.

1. **Potenciómetro**: Este componente simula el consumo de energía. Al girarlo, varía la resistencia, lo que cambia el valor analógico leído por el microcontrolador, simulando diferentes niveles de consumo eléctrico.

Figura 5: Potenciómetro

2. **Buzzer**: El buzzer actúa como una alerta sonora que se activa cuando el consumo de energía supera un umbral predeterminado. Si se detecta un alto consumo, el buzzer emite un sonido.

Figura 6: Buzzer

3. **LED Verde**: Indica un consumo bajo o normal de energía. Cuando el consumo está por debajo del umbral, este LED permanece encendido.

Figura 7: LED Verde

4. **LED Rojo**: Este LED se enciende cuando el consumo de energía es alto, superando el umbral establecido, y sirve como alerta visual de consumo elevado.

Figura 8: LED rojo

Estos componentes se conectan a un microcontrolador, que monitorea el valor del potenciómetro y controla los LEDs y el buzzer para mostrar alertas en función del consumo. Además, puede incluirse una pantalla LCD para visualizar los valores de consumo en tiempo real (aunque en este caso está comentado en el código).

Desarrollo del código:

Estructura básica del código:

Declaración de librerías y variables:

- Se incluye la librería LiquidCrystal.h, que se puede descomentar si se utiliza una pantalla LCD.
- Se definen los pines del potenciómetro, el buzzer, el LED verde y el LED rojo.
- Se establece un umbral de consumo (512) que determinará cuándo activar las alertas.

```
// Incluye las librerías necesarias
#include <LiquidCrystal.h> // Descomenta si estás usando una pantalla
LCD

// Definiciones de pines
const int potPin = A0;       // Pin del potenciómetro (simulación de
consumo)
const int buzzerPin = 5;     // Pin del buzzer
const int ledGreenPin = 3;   // Pin del LED verde (bajo consumo)
const int ledRedPin = 4;     // Pin del LED rojo (alto consumo)

// Umbral de consumo (ajustar según necesidad)
const float consumptionThreshold = 512.0; // Umbral de alerta (escala
del potenciómetro)
```

Configuración de pines y componentes (setup):

- En la función setup(), se inicializa la comunicación serial para monitorear el consumo simulado en el monitor serial.
- Se configuran los pines de salida para el buzzer y los LEDs.
- Si se utiliza una pantalla LCD, se puede inicializarla aquí con lcd.begin(16, 2) para un display de 16x2.

```
void setup() {
  Serial.begin(9600);

  // Configura pines
  pinMode(buzzerPin, OUTPUT);
  pinMode(ledGreenPin, OUTPUT);
  pinMode(ledRedPin, OUTPUT);

  // Configura la pantalla LCD
  //lcd.begin(16, 2); // Descomenta si estás usando una pantalla
LCD
}
```

Ciclo principal (loop):

- En la función loop(), se lee el valor del potenciómetro conectado al pin A0. Este valor simula el consumo de energía (en una escala de 0 a 1023).
- Se imprime el valor del consumo simulado en el monitor serial.
- Si se usa una pantalla LCD, se muestra el valor del potenciómetro en la pantalla.
- Basado en el valor del potenciómetro:
    - Si supera el umbral de consumo (512), se enciende el buzzer y el LED rojo, y se apaga el LED verde (indica un alto consumo).
    - Si está por debajo del umbral, el buzzer y el LED rojo se apagan, y se enciende el LED verde (indica bajo consumo).
- El sistema espera 1 segundo antes de hacer una nueva lectura y controlar las alertas nuevamente

```
void loop() {
  // Lee el valor del potenciómetro
  int potValue = analogRead(potPin);

  // Muestra el valor en el monitor serial
  Serial.print("Consumo simulado: ");
  Serial.print(potValue);
  Serial.println(" (escala 0-1023)");

  // Muestra los datos en la pantalla LCD (descomenta si se usa)
  //lcd.clear();
  //lcd.setCursor(0, 0);
  //lcd.print("Consumo: ");
  //lcd.print(potValue);
  //lcd.print(" (escala 0-1023)");

  // Control de alertas
  if (potValue > consumptionThreshold) {
    digitalWrite(buzzerPin, HIGH); // Enciende el buzzer
    digitalWrite(ledRedPin, HIGH); // Enciende el LED rojo
    digitalWrite(ledGreenPin, LOW); // Apaga el LED verde
  } else {
    digitalWrite(buzzerPin, LOW); // Apaga el buzzer
    digitalWrite(ledRedPin, LOW); // Apaga el LED rojo
    digitalWrite(ledGreenPin, HIGH); // Enciende el LED verde
  }

  delay(1000); // Espera 1 segundo antes de la siguiente lectura
}
```

DATOS ÚTILES

Puedes utilizar distintos simuladores, tanto gratuitos como de pago, para aprender y profundizar en el funcionamiento de los circuitos. Cada simulador ofrece herramientas únicas que te ayudarán a simular, analizar y comprender mejor el comportamiento de los componentes electrónicos.

Con esta estructura básica en mente, podrás implementar cualquier proyecto de electrónica y adaptar tu código según el simulador que estés utilizando, ya sea Tinkercad, Proteus o Wokwi.

Conexión de componentes:

Aquí te proporcionamos una guía sobre cómo deben conectarse. Sigue los mismos pasos si vas a utilizar el código proporcionado, y si cambias algún pin, no olvides hacer el ajuste correspondiente en el código.

Componentes	Abreviatura
Potenciómetro	Pot1
Buzzer	Bz1
LED(verde, rojo)	Led1, led2
Resistencias	r1, r2

COMPRUEBA TU APRENDIZAJE

Experimentar con diferentes valores de resistencias en tu simulación te permitirá observar cómo varía el comportamiento del circuito, optimizando su rendimiento y eficiencia.

DATOS ÚTILES

Multisim: Ofrece herramientas avanzadas para análisis y simulación de circuitos analógicos y digitales, ideal para educación y diseño profesional. Su interfaz es amigable y permite realizar simulaciones en tiempo real.

Diagrama esquemático:

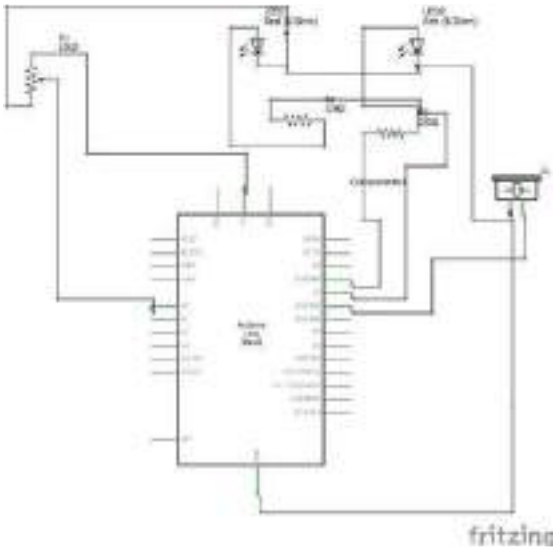


Figura 9: diagrama esquemático del circuito proporcionado por el software fritzing

PROBLEMAS DE SISTEMAS ELECTRÓNICOS

SISTEMA DE RIEGO INTELIGENTE CON ARDUINO UNO Y SENSORES DE HUMEDAD: AUTOMÁTICO, BASADO EN LA DETECCIÓN DE HUMEDAD DEL SUELO

Un **Sistema de Riego Inteligente con Arduino Uno y Sensores de Humedad** permite automatizar el riego de plantas de manera eficiente, basándose en la detección de los niveles de humedad del suelo. Este sistema es ideal para mantener el cuidado adecuado de jardines, huertos o cultivos sin la necesidad de intervención constante.

El sistema funciona midiendo la humedad del suelo mediante sensores conectados al Arduino Uno. Cuando se detecta que los niveles de humedad son bajos, el sistema activa automáticamente una bomba de agua o un sistema de riego, asegurando que las plantas reciban la cantidad adecuada de agua en el momento justo.

SABÍAS QUE

Los sistemas de riego inteligente pueden ahorrar hasta un 50% de agua, ya que activan el riego solo cuando el sensor de humedad indica que el suelo está seco, optimizando el uso del agua en la agricultura y la jardinería.

Además de ahorrar tiempo y esfuerzo, este sistema inteligente ayuda a evitar el desperdicio de agua y garantiza que las plantas no sufran ni por exceso ni por falta de riego, optimizando el uso de los recursos y contribuyendo al cuidado del medio ambiente.

DISEÑO DE PROTOTIPO

En este circuito, se utilizan tres componentes clave para implementar un sistema de riego automático basado en la humedad del suelo:

1. **Sensor de Humedad**: Este sensor está conectado al pin analógico del Arduino Uno. Su función es medir la humedad del suelo y enviar una señal analógica al Arduino, representando los niveles de humedad. Según esta lectura, el sistema decidirá si es necesario activar o desactivar el riego.

Figura 10: potenciómetro que simula el sensor de humedad

2. **Relé**: El relé está conectado al pin del Arduino. Su función es actuar como un interruptor controlado electrónicamente, permitiendo la conexión o desconexión de la bomba de agua al sistema. Cuando el Arduino detecta niveles bajos de humedad, activa el relé para encender la bomba.

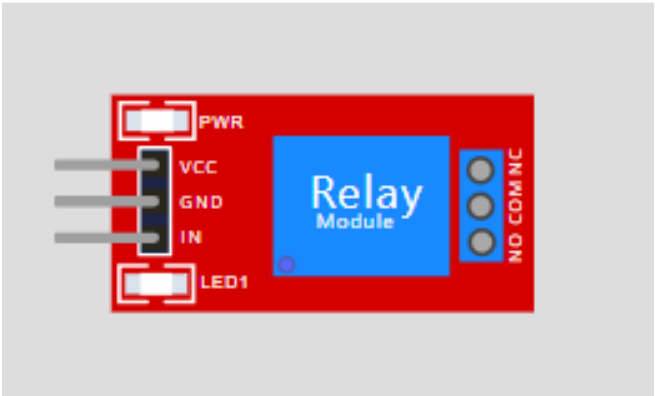


Figura 11: relé

3. **Bomba de Agua**: Cuando el relé se activa, la bomba de agua comienza a funcionar, proporcionando agua a las plantas.

Figura 12: LED roja que simula la bomba de agua

El diseño asegura que cuando el sensor detecte bajos niveles de humedad, la bomba se encienda automáticamente para regar las plantas, y se apague cuando el nivel de humedad sea suficiente.

Desarrollo del código:

Estructura del código para el Sistema de Riego Inteligente

1. **Definición de pines**:
   - Se declaran constantes para los pines donde están conectados los componentes, como el sensor de humedad y el relé.
2. **Declaración de variables**:
   - Se define una variable (humedad) para almacenar el valor leído del sensor de humedad.

```
// Definición de pines
const int sensorPin = A0;   // Pin donde se conecta el sensor de
humedad
const int relayPin = 7;     // Pin para controlar el relé
int humedad;
```

3. **Configuración inicial (función setup())**:
   - Se configuran los pines: se establece el sensorPin como entrada y el relayPin como salida.
   - Se inicializa el estado del relé para apagar la bomba al inicio.
   - Se inicia la comunicación serial para monitorear el estado de humedad.

```
// Configuración Inicial
void setup() {
  pinMode(sensorPin, INPUT);
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH);  // Apagar la bomba inicialmente
  Serial.begin(9600);  // Para monitorear la humedad en el Monitor Serie
}
```

4. **Ciclo principal (función loop())**:

   - Se lee el valor del sensor de humedad y se almacena en la variable humedad.
   - Se imprime el valor de humedad en el monitor serial.
   - Se evalúa si la humedad está por debajo de un umbral específico (400):

     - Si es así, se activa el relé para encender la bomba y se imprime un mensaje indicando que la bomba está encendida.
     - Si no, se apaga el relé, desactivando la bomba, y se imprime un mensaje indicando que la bomba está apagada.

   - Se incluye un retraso de 1 segundo entre lecturas para evitar lecturas erráticas.

```
// Bucle principal
void loop() {
  humedad = analogRead(sensorPin);  // Leer valor del sensor de humedad
  Serial.print("Humedad del suelo: ");
  Serial.println(humedad);

  // Si el suelo está seco, encender la bomba
  if (humedad < 400) {  // Ajusta el valor según el tipo de sensor
    digitalWrite(relayPin, LOW);  // Encender la bomba (activar relé)
    Serial.println("Bomba ENCENDIDA");
  } else {
    // Si el suelo está húmedo, apagar la bomba
    digitalWrite(relayPin, HIGH);  // Apagar la bomba (desactivar relé)
    Serial.println("Bomba APAGADA");
  }

  delay(1000);  // Pausa de 1 segundo entre lecturas
}
```

Conexión de componentes:

Aquí te proporcionamos una guía sobre cómo deben conectarse. Sigue los mismos pasos si vas a utilizar el código proporcionado, y si cambias algún pin, no olvides hacer el ajuste correspondiente en el código.

Componentes	Abreviatura
Potenciómetro	Pot1
Relé	Bz1
LED (rojo)	Led1, led2

Pot1: GND conectado a uno: GND.2
Pot1: SIG conectado a uno: A0
Pot1: VCC conectado a uno 5V
Relay1: VCC conectado a uno:5V
Relay1: GND conectado a uno: GND.2
Relay1:IN conectado a uno:7
Led1:C conectado a uno: GND.2
Led1: A conectado a relay1:COM

DATOS ÚTILES

Proteus: Ideal para simular circuitos electrónicos y microcontroladores, permite visualizar tanto el esquema como el diseño PCB. Es útil para validar el comportamiento del hardware antes de la implementación física.

Diagrama esquemático:

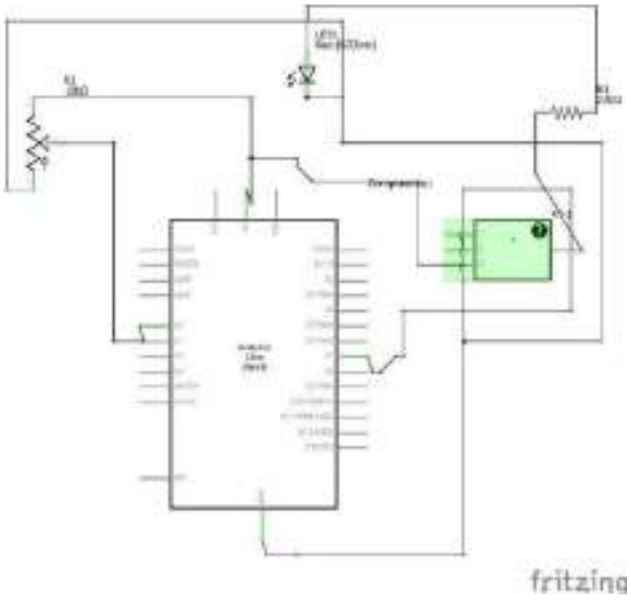


Figura 13: diagrama esquemático del circuito proporcionado por el software fritzing

PROBLEMAS DE SISTEMAS ELECTRÓNICOS

SISTEMA DE ESTACIONAMIENTO INTELIGENTE CON DETECCIÓN DE ESPACIOS LIBRES: USANDO SENSORES ULTRASÓNICOS PARA CONTROLAR UNA BARRERA Y MOSTRAR ESPACIOS DISPONIBLES

El **Sistema de Estacionamiento Inteligente con Detección de Espacios Libres** es una solución moderna que optimiza el proceso de aparcamiento, facilitando la identificación de espacios vacíos mediante el uso de sensores ultrasónicos. Este sistema detecta la presencia de vehículos en tiempo real y controla una barrera automática que se abre o cierra dependiendo de la disponibilidad de plazas.

Además de la automatización de la barrera, el sistema incluye una pantalla o indicador que muestra la cantidad de espacios disponibles, ayudando a los conductores a encontrar aparcamiento de manera más eficiente. El uso de sensores ultrasónicos permite una detección precisa de vehículos, lo que reduce el margen de error y mejora la experiencia del usuario.

SABÍAS QUE

Los sistemas de estacionamiento inteligente utilizan sensores ultrasónicos, que funcionan de manera similar a cómo los murciélagos se orientan en la oscuridad, emitiendo ondas que rebotan en los objetos cercanos para detectar espacios disponibles.

Este proyecto es especialmente útil en aparcamientos de gran afluencia, como centros comerciales o edificios corporativos, donde la gestión eficiente de espacios puede ahorrar tiempo y mejorar la fluidez del tráfico dentro del recinto.

DISEÑO DE PROTOTIPO

El diseño del circuito para este sistema de estacionamiento utiliza varios componentes electrónicos conectados al Arduino Uno. Cada componente tiene un rol específico en la detección y control del sistema:

1. **Sensores ultrasónicos**:
   - Se conectan dos sensores ultrasónicos, uno para la detección de la entrada del vehículo y otro para la detección de la disponibilidad del espacio.
   - Los pines **trig** y **echo** del sensor de entrada están conectados a los pines 8 y 9 del Arduino, respectivamente, mientras que los pines **trig** y **echo** del sensor de espacio están conectados a los pines 6 y 7.
   - Estos sensores miden la distancia entre el sensor y los objetos cercanos (como un coche), proporcionando la información para abrir la barrera o determinar si un espacio está libre.

Figura 14: Sensores ultrasónicos

2. **Servomotor**:

   - El servomotor, que controla la barrera del estacionamiento, está conectado al pin 10 del Arduino. Este motor mueve la barrera hacia arriba o hacia abajo dependiendo de si el sensor de entrada detecta un coche y si hay un espacio disponible.
   - Cuando la barrera está abierta, el servomotor se mueve a un ángulo de 90 grados, y cuando está cerrada, regresa a 0 grados.

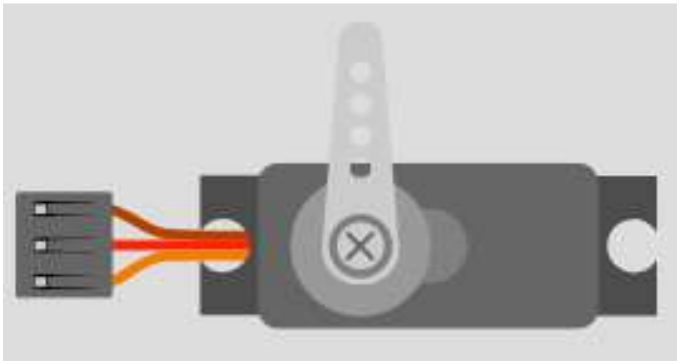


Figura 15: Servomotor

3. **LEDs**:

   - Dos LEDs indican el estado del espacio de estacionamiento: un LED verde conectado al pin 11 y un LED rojo conectado al pin 12.
   - El LED verde se enciende cuando hay un espacio libre y el LED rojo cuando el espacio está ocupado, proporcionando una señal visual clara para el conductor.

Figura 16: LED puedes ser de varios colores

Este circuito controla automáticamente el acceso de los vehículos al estacionamiento, utilizando sensores para detectar la proximidad y disponibilidad de espacios, y el servomotor para operar la barrera.

Desarrollo del código:

Estructura básica del código:

1. **Declaración de librerías y variables**:
   - Se incluye la librería Servo.h para controlar el servomotor.
   - Se declaran las constantes y variables que se utilizarán para definir los pines de los sensores ultrasónicos, servomotor y LEDs.
   - Se establecen los umbrales de distancia y los ángulos de apertura y cierre de la barrera, así como el estado inicial de la barrera (cerrada).

```
#include <Servo.h>

const int trigPinEntrada = 8;  // Pin de disparo del sensor ultrasónico en la
entrada
const int echoPinEntrada = 9;  // Pin de eco del sensor ultrasónico en la
entrada
const int trigPinEspacio = 6;  // Pin de disparo del sensor ultrasónico en el
espacio
const int echoPinEspacio = 7;  // Pin de eco del sensor ultrasónico en el
espacio
const int servoPin = 10;       // Pin del servomotor
const int ledVerde = 11;       // Pin del LED verde (disponible)
const int ledRojo = 12;        // Pin del LED rojo (ocupado)

const int umbralDistancia = 10; // Distancia en cm para abrir la barrera
const int anguloAbierto = 90;   // Ángulo de la barrera abierta
const int anguloCerrado = 0;    // Ángulo de la barrera cerrada

Servo myServo;
bool barreraAbierta = false; // Estado de la barrera
```

2. **Configuración de pines y componentes (setup)**:
   - En la función setup(), se configuran los pines de los sensores ultrasónicos como entradas y salidas, los pines de los LEDs como salidas, y el pin del servomotor.

```
void setup() {
  pinMode(trigPinEntrada, OUTPUT);
  pinMode(echoPinEntrada, INPUT);
  pinMode(trigPinEspacio, OUTPUT);
  pinMode(echoPinEspacio, INPUT);
  pinMode(ledVerde, OUTPUT);
  pinMode(ledRojo, OUTPUT);
  myServo.attach(servoPin);
  myServo.write(anguloCerrado); // Inicialmente, la barrera está cerrada
  digitalWrite(ledVerde, LOW); // LED verde apagado
  digitalWrite(ledRojo, LOW);  // LED rojo apagado
  Serial.begin(9600);
}
```

- Se inicializa el servomotor en la posición de barrera cerrada (0 grados), y se apagan ambos LEDs.
- También se inicia la comunicación serial para monitorear las distancias leídas por los sensores.

3. **Ciclo principal (loop)**:

- La función loop() es responsable de medir continuamente las distancias de los sensores de entrada y espacio utilizando la función leerDistancia().
- Dependiendo de la distancia del sensor que monitorea el espacio de estacionamiento, se enciende el LED rojo (si el espacio está ocupado) o el LED verde (si está disponible).
- Si un vehículo se acerca a la entrada y hay espacio disponible, se abre la barrera mediante la función abrirBarrera(). Si no hay un vehículo en la entrada, la barrera se cierra con la función cerrarBarrera().
- Se añade un retardo de 500 ms para evitar que las lecturas sean demasiado rápidas.

```
void loop() {
  int distanciaEntrada = leerDistancia(trigPinEntrada,
echoPinEntrada);
  int distanciaEspacio = leerDistancia(trigPinEspacio,
echoPinEspacio);

  Serial.print("Distancia Entrada: ");
  Serial.print(distanciaEntrada);
  Serial.print(" cm, Distancia Espacio: ");
  Serial.println(distanciaEspacio);

  // Controlar los LEDs basado en el estado del espacio
  if (distanciaEspacio < umbralDistancia) {
    digitalWrite(ledRojo, HIGH); // Enciende el LED rojo
    digitalWrite(ledVerde, LOW); // Apaga el LED verde
  } else {
    digitalWrite(ledRojo, LOW);  // Apaga el LED rojo
    digitalWrite(ledVerde, HIGH); // Enciende el LED verde
  }

  // Control de la barrera basado en la detección del sensor en la
entrada
  if (distanciaEntrada < umbralDistancia && !barreraAbierta &&
distanciaEspacio >= umbralDistancia) {
    abrirBarrera();
    barreraAbierta = true;
  } else if (distanciaEntrada >= umbralDistancia && barreraAbierta)
{
    cerrarBarrera();
    barreraAbierta = false;
  }

  delay(500); // Espera para evitar lecturas continuas
}
```

4. **Funciones auxiliares**:

   - leerDistancia(): Mide la distancia utilizando los pines del sensor ultrasónico (trig y echo), calcula el tiempo de retorno del eco y lo convierte en una distancia en centímetros.
   - abrirBarrera(): Gradualmente mueve el servomotor desde el ángulo cerrado (0 grados) hasta el ángulo abierto (90 grados).
   - cerrarBarrera(): Gradualmente mueve el servomotor desde el ángulo abierto hasta el ángulo cerrado.

```
int leerDistancia(int trigPin, int echoPin) {
  long duration;
  int distance;

  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);
  distance = (duration / 2) / 29.1;

  return distance;
}

void abrirBarrera() {
  for (int pos = anguloCerrado; pos <= anguloAbierto; pos += 1) {
    myServo.write(pos);
    delay(15); // Ajustar velocidad de apertura
  }
  Serial.println("Barrera abierta.");
}

void cerrarBarrera() {
  for (int pos = anguloAbierto; pos >= anguloCerrado; pos -= 1) {
    myServo.write(pos);
    delay(15); // Ajustar velocidad de cierre
  }
  Serial.println("Barrera cerrada.");
}
```

Esta estructura permite que el sistema de estacionamiento sea completamente automático, detectando la presencia de vehículos y controlando la barrera y los indicadores de espacio disponible.

Conexión de componentes:

Aquí te proporcionamos una guía sobre cómo deben conectarse. Sigue los mismos pasos si vas a utilizar el código proporcionado, y si cambias algún pin, no olvides hacer el ajuste correspondiente en el código.

Componentes	Abreviatura
Sensores ultrasónicos	Ultrasonic1, Ultrasonic2
Servomotor	Servo1
LED (verde, rojo)	Led1, led2
Resistencias	R2, R1

Ultrasonic1: VCC conectado a uno:5V

Ultrasonic1: TRIG conectado a uno:6

Ultrasonic1: ECHO conectado a uno:5

Ultrasonic1: GND conectado a uno: GND.2

Servo1: GND conectado a uno: GND.1

Servo1: V+ conectado a uno:5V

Servo1: PWN conectado a uno:10

Ultrasonic2: VCC conectado a uno:5V

Ultrasonic2: TRIG conectado a uno:8

Ultrasonic2: ECHO conectado a uno:7

Ultrasonic2: GND conectado a uno: GND.2

Led1: C conectado a uno: GND.1

Led1: A conectado a r2:1

r2:2 conectado a uno:12

Led2: C conectado a uno: GND.1

Led2: A conectado a r1:1

R1:2 conectado a uno:11

DATOS ÚTILES

Fritzing: Se centra en la creación de prototipos electrónicos, permitiendo a los usuarios diseñar circuitos de manera intuitiva y luego convertir esos diseños en diagramas para PCB. Es excelente para principiantes que quieren aprender sobre electrónica de forma visual.

Diagrama esquemático:

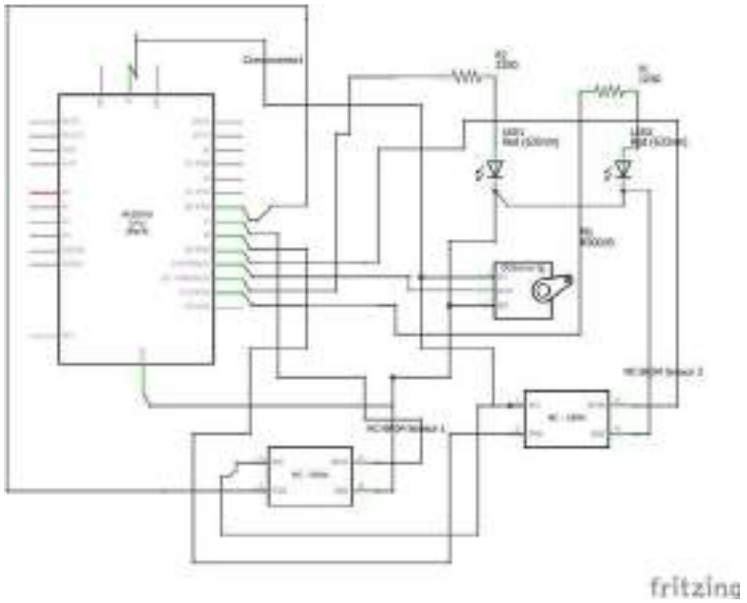


Figura 17: diagrama esquemático del circuito proporcionado por el software fritzing

Recursos

Herramientas & Software de la asignatura

Existen algunas herramientas para simular VHDL, que ya se han visto en la unidad anterior. También es posible utilizar alguna de las herramientas disponibles para diseñar circuitos digitales que se han visto previamente.

- UMHDL: IDE educacional para enseñar HDL. Funciona en varios sistemas operativos (ver wiki para instalación en Windows). Disponible en https://sourceforge.net/projects/umhdl/. Las instrucciones de uso se pueden ver en https://www.youtube.com/watch?v=7rOPzh3DO_A
- EDA playground: Entorno en línea para programación HDL. Soporta varios lenguajes (VHDL, Verilog, SystemC, SystemVerilog, entre otros). Disponible en https://edaplayground.com.
- GHDL: Un simulador VHDL open-source. Preferible instalarlo en entornos Linux, también funciona para Windows pero requiere algunos pasos adicionales. Disponible en http://ghdl.free.fr/. Se puede ver cómo usarlo en https://www.youtube.com/watch?v=5shVE94I3io.
- Logisim, herramienta para simular circuitos lógicos digitales. Disponible en https://sourceforge.net/projects/circuit/. Información sobre la herramienta en http://www.cburch.com/logisim/es/index.html.
- Logic.ly, simulador de electrónica digital en línea, aunque también hay la versión para escritorio, disponible en https://logic.ly/demo/, para aprender a utilizar este simulador se puede ver: https://www.youtube.com/watch?v=igc9HoDZl54.

Bibliografía

Brown, S., & Vranesic, Z. (2006). *Fundamentos de lógica digital con diseño VHDL* (Segunda ed.). McGraw-Hill.

Floyd, T. (2006). *Fundamentos de sistemas digitales* (Novena ed.). Pearson Educación.

Machado, F., & Borromero, S. (2010). *Diseño de circuitos digitales con VHDL* (Primera ed.). Universidad Rey Juan Carlos.

Morris Mano, M. (2003). *Diseño Digital* (Tercera ed.). Pearson Educación.

INFORME DE AUTENTICIDAD TURNITIN

CÁMARA ECUATORIANA DEL LIBRO

FICHA REGISTRO DE ISBN
INTERNATIONAL STANDARD BOOK NUMBER
Agencia ISBN Ecuador
Cámara Ecuatoriana del Libro
Eloy Alfaro N29-61 e Inglaterra, 3° Piso.
http://www.celibro.org.ec

No Radicación 167089

Fecha de asignación: 2024-11-12

Tipo de Obra	Información del Título
ISBN Obra Independiente: 978-9942-7225-2-3	Título: Electrónica y automatización de 0 99
ISBN Volumen:	Título:
ISBN Obra Completa:	Título:
Sello editorial: Instituto de Investigación Multidisciplinaria Perspectivas Globales (978-9942-7225)	

Subtítulo
Subtítulo Obra Independiente:
Subtítulo Obra Volumen:
Subtítulo Obra Completa:

Tema		
Materia: 537 - Electricidad y electrónica		Tipo de Contenido: Libros universitarios
CLASIFICACIÓN THEMA		
TJ - Ingeniería electrónica y de las comunicaciones		
Colección: Tecnología	No colección: 3	Serie: Tecnología
Público objetivo: Enseñanza universitaria o superior.		
IDIOMAS		
Español		

Colaboradores y Autor(es)		
Nombre	**Nacionalidad**	**Rol**
Félix López, Miryam Elizabeth	Ecuador	Autor
Moreira Zambrano, Cesar Armando	Ecuador	Autor
Vera Vera, José Belisario	Ecuador	Autor
Carrera Sánchez, Freddy Andrés	Ecuador	Autor
Loor Vera, Yimmy Salvador	Ecuador	Autor
Cedeño Cedeño, Ángel Reinaldo	Ecuador	Autor
Vargara Cedeño, Alexis Xavier	Ecuador	Autor
Burgos Briones, José Galdino (Licenciado)	Ecuador	Coordinador Editorial
Aldana Zavala, julio juvenal	Venezuela	Director

Traducción			
Traducción: No	Del:	Al:	Idioma Original:
Título Original:			

Información de Edición			
No de Edición: 1	Ciudad de Edición: Portoviejo	Departamento, Estado o Provincia: Manabí	Fecha de aparición: 2024-11-18
Coedición: No		Coeditor:	

Comercializable	
No de ejemplares oferta nacional: 20	Precio en moneda local:
No de ejemplares oferta externa: 0	Precio en dólares:
Oferta total: 20	
Disponibilidad: Disponible	Estatus en el catálogo: Próxima aparición

Descripción física - Impresión en papel			
Descripción física: Libro	No páginas: 224	Tipo de impresión: Tipográfico	No tintas: 1
Tipo de encuadernación: Tapa blanda o rústica	Tipo papel: Papel bond o papel obra		Gramaje: 0-19
Tamaño: 21x22	Peso: 40 g.		

Descripción física - Medio electrónico o digital

CÁMARA ECUATORIANA DEL LIBRO

FICHA REGISTRO DE ISBN
INTERNATIONAL STANDARD BOOK NUMBER
Agencia ISBN Ecuador
Cámara Ecuatoriana del Libro
Eloy Alfaro N29-61 e Inglaterra, 5° Piso.
http://www.celibro.org.ec

No Radicación 167089

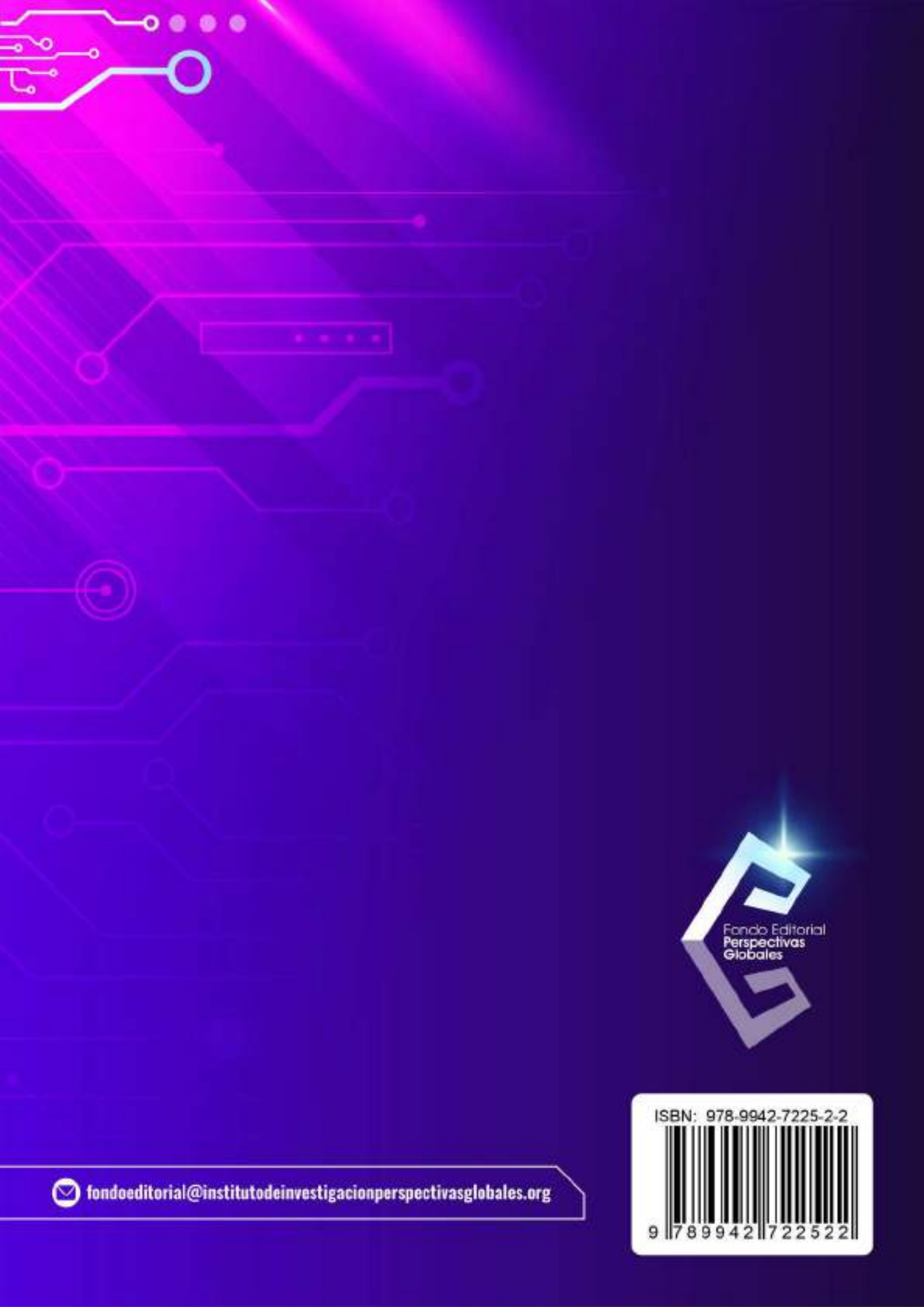
Fecha de asignación: 2024-11-12

Tipo de soporte:	Formato:	Tipo de contenido:
Medio electrónico o digital:	Protección técnica:	Permiso de uso:
Tipo de restricción de uso:	Tipos de acceso:	Tamaño:

Editorial: Instituto de Investigacion Multidisciplinario Perspectivas Globales		
Número de identificación tributaria o de ciudadanía: 1381910005001		Teléfono: 0998337081
Representante legal: Julio Juvenal Aldana Zavala		
Responsable ISBN: Julio Juvenal Aldana Zavala	e-mail: [redacted]	Celular: 0998337081

Control de Agencia


Fondo Editorial
Perspectivas
Globales



ISBN: 978-9942-7225-2-2

fondoeditorial@institutodeinvestigacionperspectivasglobales.org