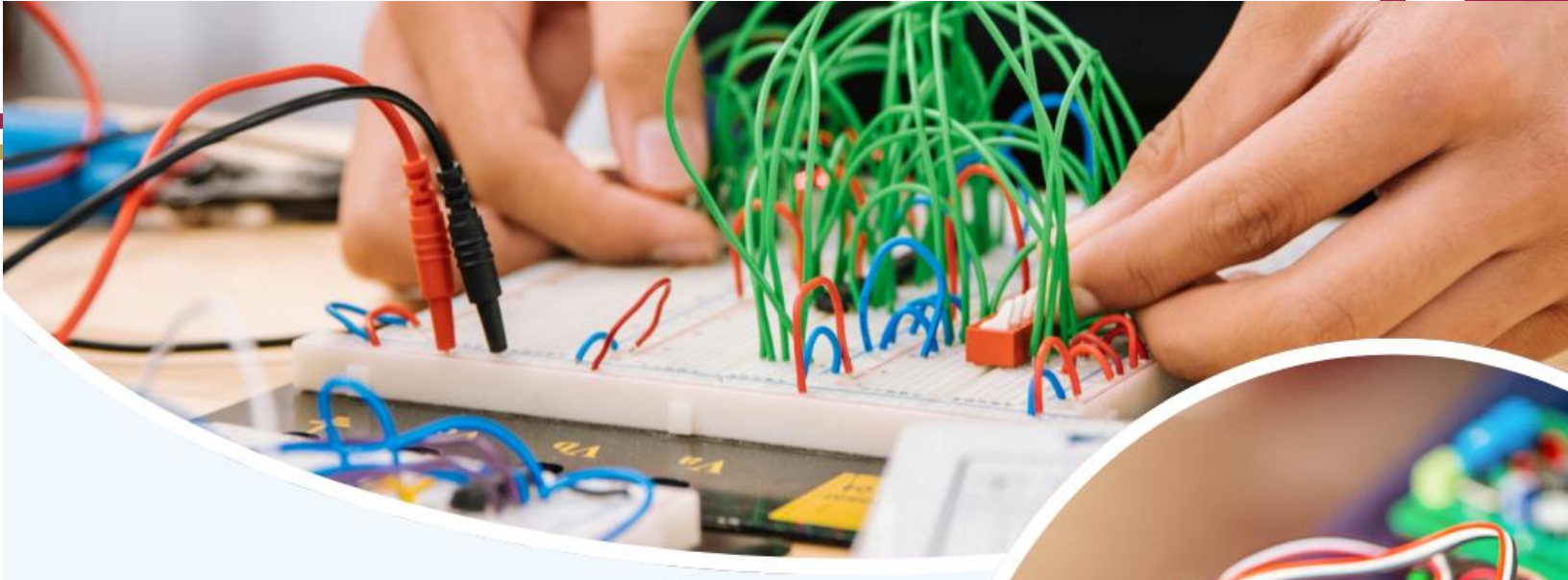
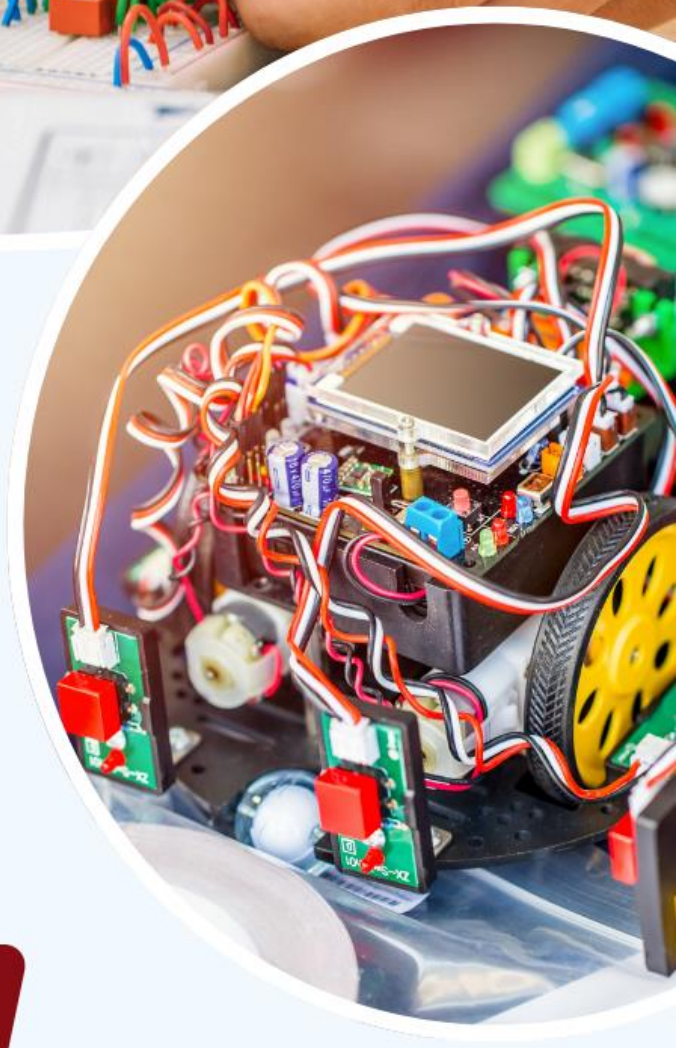




MODELO ACADÉMICO DE LA FORMACIÓN LABORAL BÁSICA: ROBÓTICA

Formación Laboral Básica
COBATAB

Módulo III: Robótica Industrial

Submódulos:

- ☑ I. Microcontroladores y Microprocesadores
- ☑ II. Control Industrial

Quinto Semestre
2026 – 2027A

MODELO ACADÉMICO ESTUDIANTE

DATOS DEL DOCENTE

Nombre: _____

Plantel _____ **Grupo** _____ **Turno** _____

Directorio

COLEGIO DE BACHILLERES DE TABASCO

Mtro. Fernando Yrys Hernández
Director General

Mtra. Ana Fanny Hernández Montero
Directora Académica

Lic. Celis del Carmen Ranero Lara
Subdirectora de Planeación Académica

Mtra. Natividad Vertiz Hernández
Jefa del Departamento Capacitación para el Trabajo

FORMACIÓN LABORAL BÁSICA: ROBÓTICA

MÓDULO III. ROBÓTICA INDUSTRIAL
SUBMÓDULO I. "MICROCONTROLADORES Y MICROPROCESADORES"
SUBMÓDULO II. "CONTROL INDUSTRIAL"

En la realización del presente material participaron:

Coordinador: Dra. Nathalia Rivera Rodríguez
Asesor: Dra. Nathalia Rivera Rodríguez
Participante: Dra. Nathalia Rivera Rodríguez

Edición: 2026-2027A

Revisado por moderador a cargo del seguimiento académico:

Lic. Martín Ricardo Beltrán Cháirez

Este material fue elaborado bajo la coordinación y supervisión del Departamento Capacitación para el Trabajo de la Dirección Académica del Colegio de Bachilleres de Tabasco y el contenido del mismo es gratuito, sin fines de lucro y con la responsabilidad de su buen uso para fines educativos www.cobatab.edu.mx

CONTENIDO

GUÍA DE ESTUDIANTES

Estimada comunidad estudiantil del Colegio de Bachilleres de Tabasco: quiero hacer énfasis en nuestra Institución, la cual tiene 5 décadas forjando el aprendizaje de cada uno de los jóvenes, hoy muchos de los que han pasado por nuestras aulas de clases son profesionistas exitosos.

Dentro del marco del 50 aniversario COBATAB, me permito como Director General enviarles un cordial saludo y dirigirme a ustedes a través de esta guía de estudio diseñada para acompañarlos como una herramienta en su formación académica y personal.

En esta etapa de su vida, el aprendizaje autónomo es clave para descubrir sus talentos, fortalecer sus habilidades y construir un futuro con propósito. Cada actividad, cada lectura y cada reto que encuentren en esta guía de estudio está pensado para impulsar su Desarrollo integral.

Les invito a asumir con responsabilidad su proceso de aprendizaje, a colaborar con sus compañeros, docentes y a mantener siempre una actitud de respeto y apertura. En el Colegio de Bachilleres de Tabasco promovemos la equidad y creemos firmemente en el potencial de cada uno de ustedes.

Esta guía de estudio busca ser clara y accesible, con objetivos definidos y contenidos relevantes para su contexto. Les animo a aprovecharla al máximo, a preguntar, reflexionar y aplicar lo aprendido en su vida diaria.

Confío en su capacidad para superar desafíos, alcanzar metas y contribuir positivamente a su comunidad. Cuentan con el apoyo de sus docentes y de todos los que integramos esta institución.

Con entusiasmo y compromiso, sigamos construyendo juntos una educación de calidad, que genera cambio.

Con aprecio y confianza,

Fernando Yrys Hernández
Director General
Colegio de Bachilleres de Tabasco

"Educación que genera cambio"

Presentación

Desde su creación, el Colegio de Bachilleres de Tabasco tiene como objetivo impartir educación de nivel medio superior en la modalidad de bachillerato general, operando desde el año 2009 un plan de estudios bajo un enfoque por competencias a través del Marco Curricular Común definiendo así los rasgos del perfil del egresado de la EMS en donde las competencias genéricas en su conjunto delinean el perfil deseable del joven bachiller.

En este sentido, los planteamientos del Marco Curricular Común de la Educación Media Superior (MCCEMS) 1 buscan una formación integral en el estudiantado mediante el desarrollo de la capacidad creadora, productiva, libre y digna del ser humano, con amor al país, a su cultura e historia. Por ello, el Bachillerato General plantea las diversas Unidades de Aprendizaje Curricular (UAC) y Formación Laboral para el Trabajo para que con sus estudiantes egresados contribuya al logro de su objetivo específico, el cual radica en la "conformación de una ciudadanía reflexiva, con capacidad de formular y asumir responsabilidades de manera comunitaria, interactuar en contextos plurales y propositivos, trazarse metas y aprender de manera El Colegio de Bachilleres de Tabasco a través de su Componente de Formación Laboral Básica ofrece en diez de sus planteles la capacitación de Robótica con la cual se busca desarrollar en el estudiantado competencias laborales básicas, que les permitan aplicar en forma integrada los conocimientos, destrezas, habilidades, actitudes y valores con responsabilidad y autonomía para desenvolverse en contextos específicos del desarrollo personal, académico, social y profesional en situaciones de la vida común, de estudio o trabajo a lo largo de la vida.1

En ese contexto, conforme a las necesidades sociales, económicas y demográficas en relación y congruencia con los objetivos y líneas de acción del programa institucional para el Colegio de Bachilleres de Tabasco, se presenta la formación laboral básica en Robótica, especifica del bachillerato general, con objetivos delimitados acorde a las características del subsistema y la necesidad del

conglomerado social en el cual será aplicado, situación que deviene de tendencias académicas que obligan a este subsistema a tener en consideración las áreas de conocimiento y perfil profesional y su estrecha relación con la demanda ocupacional y la oferta laboral, esto con plena convicción de que los estudiantes del Colegio de Bachilleres de Tabasco concluyan satisfactoriamente con sus estudios de educación media superior e ingresen a sus estudios de nivel superior con herramientas y conocimientos que son de interés general conforme al área geográfica y entorno social en el que se desarrollen; por lo que el presente documento se encuentra conformado en apartados mediante los cuales describe la justificación y los elementos claves para su implementación en el aula. continua y colaborativa".

1 El cual puede ser consultado a través del siguiente enlace: https://educacionmediasuperior.sep.gob.mx/work/models/sems/Resource/13516/1/images/Documento%20base%20MCCEMS.pdf

Programa de estudios de la formación laboral básica

Semestre	Tercero, Cuarto, Quinto y Sexto	
Créditos	56 totales/14 por semestre	
Componente	De formación laboral	
Nivel de formación laboral	Básica	
Tiempo asignado	**Mediación docente**	**Estudio independiente**
	448 h. totales 112 h. por semestre	112 h. totales 28 h. por Semestre
Sector productivo	Servicios educativos [2]	

[2] Acorde al RENEC – Registro Nacional de Estándares de Competencia por Sector Productivo. Disponible en https://conocer.gob.nx/renec-registro-nacional-de-estandares-de-competencia-porsector-productivo/

Fundamentación

La Dirección General del Bachillerato (DGB), acorde a la Nueva Escuela Mexicana (NEM) y al Marco Curricular Común de la Educación Media Superior (MCCEMS), y en su responsabilidad de enfrentar tanto los nuevos retos como los objetivos que de estos se desprenden, actualiza el presente Programa de estudios, el cual responde a una visión de educación integral, pertinente, de calidad y excelencia.

Dicho programa forma parte del Componente de Formación Laboral Básica del Bachillerato General, el cual busca ser un espacio vinculado con el sector productivo, permitiendo al estudiantado no solo cumplir con su trayecto educativo, sino construir su proyecto de vida, con mayores posibilidades de inserción en el mercado de trabajo.

Esto atendiendo al mandato constitucional que en materia educativa, con base en la Reforma Constitucional a los artículos 3°, 31° y 73° de la Constitución Política de los Estados Unidos Mexicanos y de la emisión de la Legislación Secundaria, publicadas el 30 de septiembre de 2019 en el Diario Oficial de la Federación, determinan la reorientación del Sistema Educativo Nacional para "garantizar el derecho a la educación con un enfoque de derechos humanos y de igualdad sustantiva, para incidir en la cultura educativa mediante la corresponsabilidad y el impulso de transformaciones sociales dentro de la escuela y en la comunidad".

Bajo este contexto, es que el Componente de Formación Laboral Básica, adquiere mayor relevancia, pues tendrá como ejes rectores:

- Enfoque en competencias, que busca desarrollar las capacidades y habilidades necesarias para el desempeño laboral.
- Enfoque humanista, que valora y respeta la diversidad y la dignidad del estudiantado, su potencial creativo, su participación, su bienestar integral y su compromiso social.

Estos enfoques se orientan a promover una educación inclusiva, equitativa y de calidad, que favorezca el aprendizaje permanente y el máximo logro de los aprendizajes.

Así pues, el Componente de Formación Laboral Básica, busca desarrollar en el estudiantado competencias laborales básicas, que les permitan aplicar en forma integrada los conocimientos, destrezas, habilidades, actitudes y valores con responsabilidad y autonomía para desenvolverse en contextos específicos del desarrollo personal, académico, social y profesional en situaciones de la vida común, de estudio o trabajo a lo largo de la vida.3

Para lograr dicho propósito, lo que a continuación se presenta es una serie de competencias laborales básicas las cuales permitirán al personal docente el abordaje de aprendizajes con la amplitud y profundidad acorde a los diversos contextos.

Es decir, a partir de estas competencias , se diseñarán estrategias de enseñanza-aprendizaje que permita a las y los estudiantes ser capaces de conducir su vida hacia su futuro con bienestar y satisfacción, con sentido de pertenencia social, conscientes de los problemas sociales, económicos y políticos que aquejan al país, pero también de su entorno inmediato, dispuestos a participar de manera responsable y decidida en los procesos de democracia participativa y a comprometerse en las soluciones de las problemáticas que los aquejan y que tengan la capacidad de aprender a aprender en el trayecto de su vida. 4

Para lo cual, es de primordial importancia visualizar que debe existir una articulación contextualizada del Componente Laboral Básico, con el Currículum Fundamental y el Ampliado, para garantizar así la transferencia de los conocimientos, experiencias, habilidades, capacidades, actitudes y valores.

[3] Subsecretaría de Educación Media Superior. (2023b). El currículum laboral en la educación media superior. SEP.

Es decir, esta articulación permitirá: La transversalidad del conocimiento adquirido en el Currículum Fundamental con las competencias laborales básicas requeridas en el mercado laboral.

- Fomentar el aprendizaje en contextos diversos, utilizando métodos y estrategias de aprendizaje.
- Centrar las necesidades del mercado laboral.
- Fomentar la interacción entre la vida educativa y la comunidad, a fin de propiciar la adquisición de conocimientos y competencias, de acuerdo con el desarrollo biopsicosociocultural del estudiantado.
- Aprovechar, mediante el Programa Aula, Escuela, Comunidad (PAEC), todas las oportunidades para poner en práctica lo que se ha aprendido; su aplicación no se limitará solo a los recursos sociocognitivos y a las áreas de conocimiento, sino que abarcará los aspectos funcionales (competencias laborales) y recursos socioemocionales.
- Mejorar la relación entre la escuela y los sectores productivos.5

[4] Subsecretaría de Educación Media Superior. (2023b). El currículum laboral en la educación media superior. SEP.

[5] Subsecretaría de Educación Media Superior. (2023b). El currículum laboral en la educación media superior. SEP.

Justificación de la Formación Laboral Básica en Robótica

La Formación Laboral Básica en Robótica se ubica dentro del Área de Conocimiento de las Ciencias experimentales, las cuales en el MCCEMS tienen como objeto de estudio orientar el aprendizaje de las y los estudiantes hacia una visión más científica y coherente acorde a las necesidades actuales. Se utilizan los conceptos centrales, los transversales y las prácticas de ciencia e ingeniería en forma apropiada al contexto, entendiendo la naturaleza como un fenómeno complejo y multidisciplinar, en donde se planteen situaciones que permitan comprender la forma en que la ciencia se desarrolla y se aplica en la vida diaria. Es importante trabajar colectivamente en la construcción del conocimiento, la comprensión más amplia de la forma como funciona el mundo natural y la forma en que la humanidad aprovecha este conocimiento.

Con base en ello, a lo largo del Componente de Formación Laboral Básica, se brindará una formación de carácter genérico y transversal que le permita al estudiantado incorporarse al sector productivo con actividades relativamente sencillas, de autonomía y responsabilidad mínima.
De manera específica, a través de la Formación Laboral Básica en Robótica, el estudiantado conocerá una serie de herramientas y técnicas indispensables que tienen como fin desarrollar habilidades y destrezas del área por medio de la aplicación de los principios mecánicos, eléctricos, electrónicos e informáticos que les permita obtener la formación para realizar el diseño de robots y programarlos para controlarlos, con lo cual se contribuye a la solución de su entorno y su comunidad. Así se estará vinculado de forma interdisciplinar con el campo de Matemáticas y con el campo de Física, al aportar mediante los robots la solución de diversas problemáticas.

Con base en lo anterior, esta Formación Laboral Básica tiene el propósito de: Estructura prototipos de robótica a través del diseño de robots y sus controladores, mostrando un comportamiento responsable y ético en su construcción con la finalidad de favorecer a la solución de problemáticas existentes en su entorno y

mejorar su vida cotidiana.
De esta manera, las personas egresadas de esta Formación Laboral Básica se pueden integrar a la vida laboral tanto en instituciones públicas como privadas en los siguientes perfiles ocupacionales: auxiliar en áreas de desarrollo de robots, programación, análisis de datos, electrónica, mecánica, TICS, automatización, inteligencia artificial.

La robótica en la Agenda 2030.

En el año 2015 se aprobó por parte de los Gobiernos la Agenda 2030 para erradicar la pobreza, cuidar la Tierra, mejorar la vida y las perspectivas de las personas de todo el mundo. En donde los propósitos se plasmaron en la aprobación de los 17 Objetivos de Desarrollo Sostenible.

En 15 años se busca implementar cambios que ayuden a conseguir un mundo más sostenible y en los que la robótica juegue un papel clave. De acuerdo con la International Fedration of Robotics apoya el desarrollo de los ODS y se identifican 13 de los 17 en los cuales los robots son de ayuda, ahorran recursos y contribuyen a la creación de tecnologías vedes.

En el sector agrícola, vemos como robots y cobots impulsan la agricultura inteligente, respondiendo al ODS 2: Hambre cero, eliminando los productos químicos, ayudan a localizar las malas hierbas y las pueden eliminar con rayo láser, a controlar los cultivos, su desarrollo, maduración, tamaño óptimo para recolectarlas.

Los robots contribuyen a alcanzar el ODS 3: Salud y bienestar, la medicina es un área grande para la robótica junto con la inteligencia artificial aportan un gran valor con numerosas aplicaciones que abarcan desde la cirugía, ayudan a los doctores a detectar melanomas o la fisioterapia, ayudando a recuperarse de lesiones y recopilan la evolución del paciente.

Con la salud se vincula el ODS 6: Agua limpia y Saneamiento, en donde la robótica ayuda a mejorar la red de saneamiento de aguas.

Vemos como los robots industriales y cobots forman parte de líneas de producción en grandes fábricas, respondiendo al ODS 12: Producción y consumo responsables, que están cambiando sus fuentes de energía, apostando por las limpias o renovables como la solar.

Ubicación de la Formación Laboral Básica en Robótica

1er. Semestre	2do.Semestre	3er. Semestre	4to. Semestre	5to. Semestre	6to. Semestre
Cultura Digital I	Cultura Digital II	Inglés III	Taller de Cultura Digital	UACS Optativas	UACS optativas
Inglés I	Inglés II	Pensamiento Matemático III	Temas Selectos de Matemáticas I		
Pensamiento Matemático I	Pensamiento Matemático II	Taller de Ciencias II	Inglés IV		
UACS de 1er. Semestre	Taller de Ciencias I	UACS de 3er. Semestre	UACS de 4to. Semestre	Currículum Ampliado	Currículum Ampliado
Currículum Ampliado	UACS de 2do. Semestre	Formación Laboral Básica en Robótica			

Mapa de la Formación Laboral Básica en Robótica

Modulo I: Robótica básica

Submódulo 1	Fundamentos de la robótica 48 horas de mediación docente 12 horas de estudio independiente 6 créditos
Submódulo 2	Diseño y construcción de robots 64 horas de mediación docente 16 horas de estudio independiente 8 créditos

Modulo II: Robótica educativa

Submódulo 1	Robótica pedagógica 48 horas de mediación docente 12 horas de estudio independiente 6 créditos
Submódulo 2	Robots educativos 64 horas de mediación docente 16 horas de estudio independiente 8 créditos

Modulo III: Robótica industrial

Submódulo 1	Microcontroladores y microprocesadores 48 horas de mediación docente 12 horas de estudio independiente 6 créditos
Submódulo 2	Control industrial 64 horas de mediación docente 16 horas de estudio independiente 8 créditos

Módulo IV: Robótica Móvil

Submódulo 1	Robots móviles 48 horas de mediación docente 12 horas de estudio independiente 6 créditos
Submódulo 2	Drones 64 horas de mediación docente 16 horas de estudio independiente 8 créditos

Competencias Laborales Básicas

Hacen referencia a la capacidad para aplicar conocimientos, destrezas, habilidades, actitudes y valores en el desarrollo personal, académico, social y profesional en situaciones de la vida común, de estudio o trabajo para que el estudiantado desarrolle la formación elemental o básica para el trabajo, que les permite desempeñar funciones laborales de nivel dos de competencia, aplicando soluciones a problemas simples en contextos conocidos y específicos.

Tienen validez oficial dentro del Sistema Educativo Nacional (SEN), lo cual se expresa con la emisión del documento que acredita su formación.

Estas competencias se caracterizan por:

- Pertinencia: Atiende a las necesidades del sector productivo y son valoradas.
- Relevancia: Favorece la empleabilidad y emprendimientos productivos, sin disparidades de género, étnicas o exclusión de grupos vulnerables.
- Coherencia: Acorde al tipo educativo de media superior.
- Suficiencia: Abarca un proceso completo e independiente a las demás funciones que desempeña quien egresa de la formación para el trabajo, técnica o tecnológica en el sector productivo.
- Autonomía: Faculta para el análisis y toma de decisiones.
- Responsabilidad: Capacidad para asumir compromisos orientados al logro de objetivos y metas laborales.
- Variedad: Abarca la ejecución de actividades rutinarias – no rutinarias, predecibles – impredecibles – contextos diversos.
- Complejidad: Moviliza recursos cognitivos, procedimentales y actitudinales en diferentes niveles para la ejecución de actividades y funciones.

De igual forma es importante señalar, que, para el caso de la Formación Laboral Básica, como se señaló anteriormente, se logrará en el estudiantado el nivel de competencia dos, el cual se caracteriza por:

- Realización de actividades programadas
- Aplicación de habilidades cognitivas y de comunicación para recibir, transmitir y recordar información
- Utiliza técnicas, materiales, herramientas y equipamiento que no requieren un nivel de especialización para realizar actividades en contextos conocidos, además del uso de tecnologías de la información y comunicación básicas, actuando con ética, con un enfoque de sostenibilidad y responsabilidad sobre su entorno.6
- Así mismo, para el caso de la Formación Laboral Básica en Robótica, se considerará el siguiente perfil de egreso.:
- Ordena propuestas de robótica básica, a través de sus fundamentos. Arquitectura y anatomía, para desarrollar soluciones a problemáticas que se plantean de la vida cotidiana, de forma creativa, responsable y organizada.
- Asocia las fases de diseño y desarrollo de un robot, identificando sus funciones y utilizando software de programación, para moverlo y controlarlo, con la finalidad de promover un pensamiento crítico, analítico, así como un trabajo metódico y organizado.
- Interpreta las fases de la robótica en el diseño y desarrollo de robots pedagógicos, reconociendo la función de cada una de ellas y usando la programación para controlarlo y moverlo, promoviendo la tolerancia a la frustración ante retos y fallas.
- Explica las fases de la robótica en el diseño y desarrollo de robots educativos, comprobando su funcionalidad y analizando las necesidades de su entorno con el fin de desarrollar un pensamiento crítico y creativo.
- Estructura con acompañamiento de su docente proyectos robóticos con microcontroladores, microprocesadores y circuitos lógicos, a partir de las necesidades existentes en la comunidad, permitiendo la solución de problemáticas de forma responsable e innovadora.
- Selecciona en compañía de su docente programas para PLC mediante la codificación y compilación de las instrucciones pertinentes para cumplir con

los requerimientos de funcionalidad y rendimiento establecidos de control industrial, actuando de forma congruente y consciente previniendo riesgos en situaciones cotidianas.

- Reconoce la importancia del comportamiento ético y responsable en el diseño de robots móviles creativos y funcionales, con la finalidad de promover una conducta socialmente benéfica en su comunidad.
- Identifica los diversos tipos de drones reconociendo sus fases, el software de programación para controlarlos, y tomando decisiones creativas y responsables en su construcción para resolver problemáticas inmersas en su contexto.

Así mismo se señalan algunas normas de competencia laboral vigentes, avaladas por el Consejo Nacional de Normalización y Certificación de Competencias Laborales, que podrán servir de guía para el desarrollo de los módulos:

- EO2732. Mantener en condiciones de operación los sistemas electrónicos analógicos.
- EO2733. Mantener en condiciones de operación los sistemas electrónicos digitales.
- EO2734. Mantener en condiciones de operación los sistemas con microprocesadores.
- EC0972. Programación del robot industrial.

[6] Subsecretaría de Educación Media Superior. (2023b). El currículum laboral en la educación media superior. SEP.

Proceso de evaluación bajo el enfoque en competencias

La evaluación por competencias es un proceso que permitirá mediante la obtención de evidencias conocer el dominio de conocimientos, habilidades y actitudes socioafectivas desarrolladas por el estudiantado.

Dicho proceso deberá ser formativo e integral, es decir, permitirá visualizar no solo el saber, saber hacer y saber ser, sino también el bagaje histórico y cultural lo cual permitirá al estudiantado la comprensión de la realidad social y laboral de los sectores y de la comunidad para a partir de ello lograr su intervención y aporte.

Para ello lo que a continuación se presentan son los principios que orientarán el proceso de evaluación:

1. Validez: debe existir correlación entre los resultados de la evaluación y los resultados esperados en situaciones laborales reales.
2. Confiabilidad: producir resultados consistentes al evaluar en momentos diferentes y en diversos contextos.
3. Accesibilidad: facilitar el acceso a cualquier persona que pueda ser capaz de demostrar el desarrollo de la competencia.
4. Comunicación: dar a conocer previamente las condiciones en que se va a evaluar, y posteriormente, los resultados mediante la retroalimentación.
5. Equidad: evitar cualquier práctica discriminatoria, es decir, el estudiantado será evaluado bajo los mismos criterios e indicadores.
6. Flexibilidad: adaptarse a diferentes modalidades y opciones de formación, así como a las características y necesidades del estudiantado.

Así pues para poder llevar a cabo lo aquí planteado, se propone la utilización de una amplia gama de instrumentos que permitan visualizar tanto el proceso como el resultado final del aprendizaje del estudiantado, entre los que se encuentran: rúbricas, pruebas de ejecución, portafolios de evidencias, diario de campo o

bitácora, organizadores gráficos, ensayos, resolución de ejercicios y problemas, exámenes o pruebas tipo saber, exposición, método de casos, proyectos y debates o discusiones dirigidas, entre otros.

De igual forma, es necesario que la evaluación contemple:

- Autoevaluación: cuando el estudiantado valora su desarrollo y la forma en que aprendió.
- Coevaluación: a través de la retroalimentación entre pares, fomentando la cooperación, la colaboración, la empatía y la crítica constructiva.
- Heteroevaluación: emitida por el personal docente en función de los conocimientos, habilidades, actitudes y valores ponderados en los instrumentos de evaluación.

Finalmente, respecto a los pasos para evaluar las competencias laborales básicas, se presenta a continuación una propuesta elaborada por la Subsecretaría de Educación Media Superior (2023b), la cual ilustra la ruta a seguir y constituye una referencia para el personal docente.

Pasos para evaluar competencias laborales

Fuente: Subsecretaría de Educación Media Superior, 2023b.

[7] Subsecretaría de Educación Media Superior. (2023). El currículum laboral en la educación media superior. SEP.

Rol del estudiantado

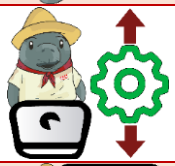
El rol del estudiantado en el proceso educativo no se limita simplemente a recibir información y repetirla, sino que debe ser un agente activo en la construcción de su propio conocimiento y de su identidad. En este sentido, no sólo se trata de aprender a leer y escribir; implica aprender a narrar y comprender su propia vida, tanto como autor o autora de su historia personal, como testigo de su contexto social y cultural. Este proceso es fundamental para que el estudiantado se convierta en un sujeto consciente y crítico de su realidad.

La educación es un motor de transformación social, pero también puede perpetuar las desigualdades existentes al tratar a todos y todas por igual sin considerar la diversidad inherente al estudiantado. La educación debe empoderarles, dándoles las condiciones necesarias para reconocer y cuestionar las desigualdades que les rodean.

Si las y los estudiantes son insertados en una educación que no considera su clase, sexo, género, etnia, lengua, cultura, capacidad, condición migratoria, religión o cualquier otro aspecto de su identidad, es muy probable que se apropien de la idea de que "la escuela no es para ellos y ellas", ya que se enfrentarían constantemente a comentarios o actitudes que les califican de incapaces, ignorantes, indolentes o inútiles terminando por creerlo y asumirlo como verdad. Esta autodesvalorización es una barrera significativa para su desarrollo ya que puede llevar a creer que el conocimiento y la sabiduría pertenecen únicamente a las y los "profesionales" y no reconocen el valor de su propio conocimiento y experiencia.

El rol de las y los estudiantes, entonces, debe ser el de un sujeto activo que desafía y transforma estas narrativas opresivas que fomentan las desigualdades. Debe aprender a valorar su propia voz y experiencia, y a reconocer su capacidad para conocer y transformar su realidad. La educación debe ser un proceso liberador que les permita verse a sí mismos o mismas como agentes de transformación social, capaces de escribir su propia historia y de participar activamente en la construcción de una sociedad más justa y humana.

Simbología

Icono	Descripción
	**Lectura**: Indica que se realizará la lectura que contiene información de los conocimientos básicos del Programa de Estudio.
	**Material audio visual**: Representa recursos adicionales al contenido de los conocimientos básicos del Programa de Estudio a través de un video mostrado por el docente, indicado por QR.
	**Actividad**: Es momento de realizar una actividad teórica – escrita que contribuye a evidenciar el propósito del aprendizaje esperado de los conocimientos básicos del Programa de Estudio.
	**Práctica**: Hagamos la práctica guiada, que contribuye a la aplicación de los saberes obtenidos con relación al conocimiento básico del Programa de Estudio, se acompaña con el instrumento de evaluación correspondiente.
	**Docente explica**: Se sugiere que esta sección el docente profundice los conocimientos para adecuarlos a su contexto.
	**Actividad SIGA**: Indica el producto que se contempla para la plataforma SIGA.
	**Evaluación Diagnóstica**: Presenta el momento para llevar a cabo la evaluación diagnóstica del submódulo.
	**Instrumento de evaluación**: Documento en el que tanto el estudiante como el docente observan los criterios con los que se evaluará el producto a realizar.
	**Encuadre**: Presenta cada aspecto y el porcentaje de calificación con el que se evaluará el submódulo.

Icono	Descripción
	**Dosificación**: Muestra las fechas en que los conocimientos básicos del Programa de Estudio se van a desarrollar.
	**PAEC:** Programa Aula, Escuela y Comunidad

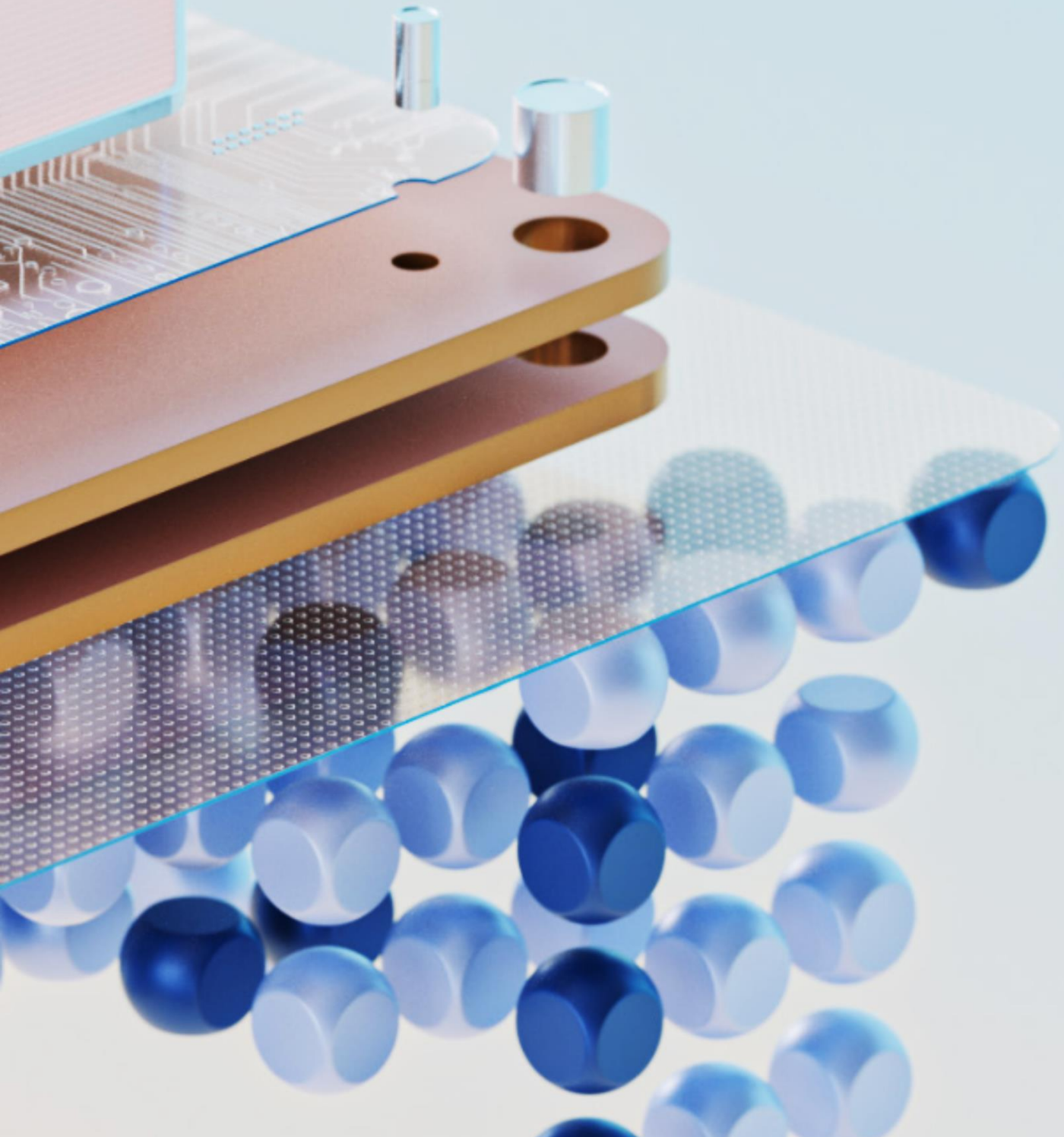
Temario

SUBMODULO: MICROCONTROLADORES Y MICROPROCESADORES

1. Microcontroladores y microprocesadores
2. Arquitectura de un microcontrolador
3. Compiladores para microcontroladores
   - 3.1 Estructura básica de un programa
   - 3.2 Tipos de datos y variables
   - 3.3 Estructuras de control
4. Configuración de puertos de entrada y salida
5. Modo de operación Stand Alone
6. Contadores
7. Interrupciones
8. Temporizadores
9. PWM
10. Conectividad serial
11. Circuitos lógicos
    - 11.1 Sistemas numéricos y conversión entre sistemas
    - 11.2 Compuertas lógicas: AND, OR, NOT, NOR, NAND
    - 11.3 Compuertas BUFFER
12. Circuitos combinacionales
13. Diagnóstico de falla de circuitos
14. Funciones de lógica combinacional

SUBMÓDULO I

MICROCONTROLADORES Y MICROPROCESADORES



Propósito del Submódulo

Plantea proyectos robóticos favoreciendo su desarrollo creativo, con microcontroladores, microprocesadores y circuitos lógicos, utilizándolos para satisfacer las necesidades existentes en su comunidad.

Sugerencias de Evidencias de Evaluación

Estructura en compañía de su docente proyectos robóticos con microcontroladores, microprocesadores y circuitos lógicos, a partir de las necesidades existentes en la comunidad, permitiendo la solución de problemáticas de forma responsable e innovadora.

Competencia Laboral Básica

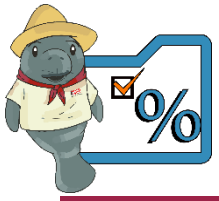
- Explica las características y alcances de microcontroladores y microprocesadores mediante diversas fuentes, favoreciendo la toma de decisiones para la solución de problemas en forma responsable y en beneficio del entorno.
- Desarrolla programas empleando adecuadamente las reglas de estructura, contenido y uso del lenguaje de programación, favoreciendo en todo momento su creatividad, siendo consciente de las consecuencias de sus actos y propositivo para la sociedad.
- Diseña proyectos con sensores y actuadores, haciendo uso de la edición, compilación y grabado de programas, empleando los puertos entrada/salida, en las partes industriales, favoreciendo su desarrollo creativo, al comportarse en forma benéfica para el bien común y afrontando las consecuencias de la toma de decisiones.
- Elabora problemas de sistemas numéricos, empleando métodos de conversión entre distintas bases, trabajando colaborativamente para comprender la forma en que la información es codificada, transmitida, procesada y almacenada.
- Construye circuitos lógicos con sistemas combinacionales por medio de la simplificación, armado y diagnóstico, en forma colaborativa, siendo creativos e innovadores para el beneficio de su comunidad y del entorno industrial.

DOSIFICACION PROGRAMATICA

Formación Laboral Básica: Robótica
Módulo III: Robótica industrial
Submódulo I: Microcontroladores y microprocesadores **Clave:** C5MM
Semestre: 5to. **Turno:** Matutino/Vespertino **Periodo:** 2026 – 2027A

Submódulo	Momento	Tiempo (minutos)	Conocimientos	Semana	Fecha inicio	Observaciones
SUBMÓDULO I. MICROCONTROLADORES Y MICROPROCESADORES	Apertura	350 min	Bienvenida Encuadre Reglamento Instrumentos de evaluación. Evaluación diagnóstica. **1. Microcontroladores y microprocesadores** **2. Arquitectura de un microcontrolador** **3. Compiladores para microcontroladores**	**1**	31 de Agosto – 04 de Septiembre de 2026	Inicio del semestre 31 de agosto
	Desarrollo	350 min	3.1 Estructura básica de un programa 3.2 Tipos de datos y variables 3.3 Estructuras de control	**2**	07 – 11 de septiembre de 2026	
		350 n	**4 Configuración de puertos de entrada y salida** 4.1 Programa: edición, compilación y guardado 4.2 Lectura y escritura en puertos **5 Modo de operación Stand Alone** **6 Contadores**	**3**	14 – 18 de septiembre de 2026	16 de septiembre suspensión oficial de labors
		350 n	**7 Interrupciones** **8 Temporizadores** **9 PWM** **10 Conectividad Serial**	**4**	21 – 25 de septiembre de 2026	
		350 n	**11 Circuitos lógicos** 11.1 Sistemas numéricos y	**5**	28 de septiembre – 02 de octubre de	

			conversiones entre sistemas 11.2 Compuertas lógicas: AND, OR, NOT, NOR, NAND 11.3 Compuertas BUFFER		2026	
		350 min	**12 Circuitos combinacionales** **13 Diagnóstico de falla de circuitos** 14 **Funciones de lógica combinacional**	**6**	05 – 09 de octubre de 2026	
	Cierre	350 min	**Situación Didáctica** "Midiendo temperatura y humedad"	**7**	12 al 16 de octubre 2026	 **Evaluación sumativa**



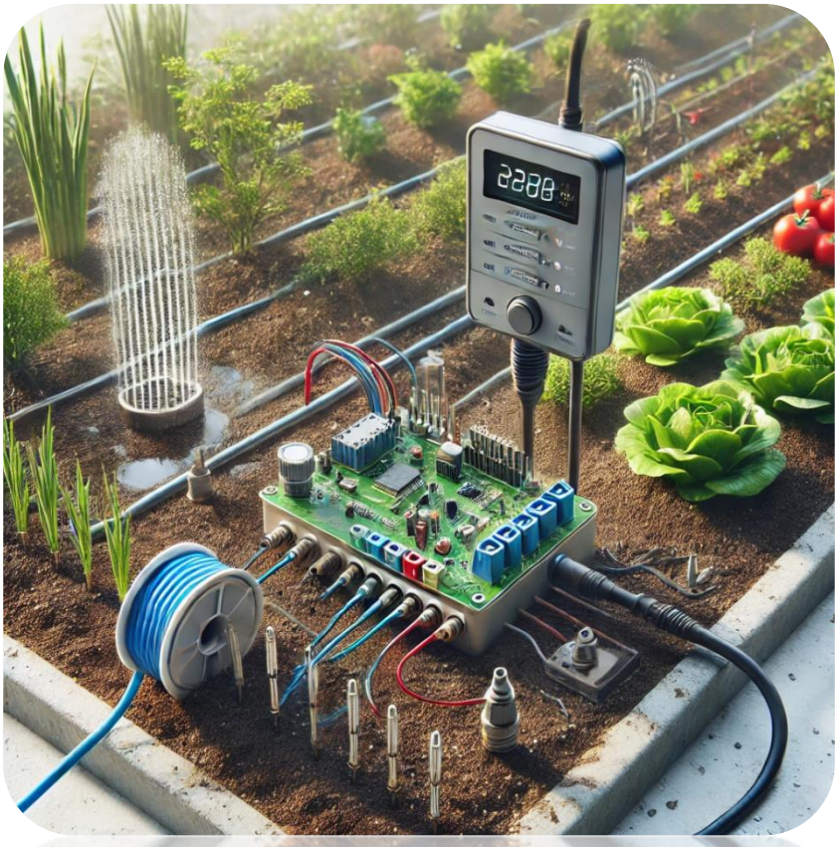
Encuadre Submódulo I Microcontroladores y Microprocesadores

Situación Didáctica	
Actividades	**Puntaje**
Actividad 1. Diferencias entre microcontroladores y microprocesadores.	3%
Actividad 2. Exposición Compiladores para microcontroladores.	3%
Actividad 3. Instalación de Arduino IDE.	3%
Actividad 4. Instrucciones de recepción serial.	3%
Actividad 5. Conversiones entre sistemas numéricos.	4%
Actividad 6. Seguimiento de señales.	4%
Suma de actividades:	**20%**
Prácticas	**Puntaje**
Práctica 1. Intermitencia de dos leds.	5%
Práctica 2. Secuencia con siete leds.	5%
Práctica 3. Secuencia con siete leds simplificado.	5%
Práctica 4. El botón del pánico.	5%
Práctica 5. Contador en display de 7 segmentos.	5%
Práctica 6. Interrupciones mediante un botón.	5%
Práctica 7. Reloj.	5%
Práctica 8. Luminosidad variable.	5%
Práctica 9. Simulación de circuitos combinacionales.	5%
Práctica 10. Decodificador.	5%
Suma de prácticas:	50%
Situación Didáctica 1. "Midiendo tempratura y humendad" 	**30%**
Total	**100%**

Situación Didáctica Submódulo I

Situación Didáctica 1	
Título:	**Midiendo temperatura y humedad**
Contexto:	La contaminación por plásticos no solo está ensuciando los océanos del mundo. También está en el aire que respiramos, viajando con el viento y descendiendo de los cielos, según un nuevo estudio. Más de 1000 toneladas de diminutos fragmentos llueven cada año sobre los parques nacionales y áreas silvestres en una diminuta parte del planeta, lo que equivale a entre 123 y 300 millones de botellas de plástico. Lo más crítico es que estudios recientes por Universidades reconocidas indican que no existe un lugar en la superficie de la tierra que no tenga microplásticos. Hoy en día, recolectar el plástico, recicla o reducir el consumo de él, es un estilo de vida común entre los jóvenes bachilleres, ya sea por moda o conciencia; aunque poco a poco se adopta una conciencia para disminuir el consumo, nos falta mucho para reducir los riesgos ocasionados por el consumo excesivo y peor aún por no generar la conciencia para reciclar o reutilizar. Sin embargo, los últimos años el avance tecnológico ha crecido gradualmente, es común que se diseñen y construyan aparatos y herramientas que clasifiquen la basura para reducir la contaminación; y que cada día un robot se encargue de hacer aquellas tareas que pudieran resultar difíciles, cansadas o muy costosas. Además, con la recolección de estos plásticos se pueden utilizar para sembrar plantas medicinales y hortalizas. Pero ¿qué pasa si no puedes comprar un aparato tan sofisticado porque sus precios son muy altos y poco accesibles? Para esto los alumnos de la Capacitación de Robótica de quinto Semestre han decidió poner en práctica lo aprendido en la Capacitación y diseñar una propuesta de robot que tenga la función de sensor de humedad y temperatura para aplicarlo en el contexto de huerto sostenible en la comunidad educativa.
Propósito	En equipo de 6 integrantes utilizando software de simulación de circuitos electrónicos y de ser posible de forma física con los materiales necesarios, realizar el Diseño, Programación, Simulación y/o puesta en marcha física de un Sensor de Temperatura y Humedad para poder ser aplicado en un contexto de huerto sostenible en la comunidad educativa.

Conflicto Cognitivo	¿Alguna vez te has preguntado cómo se llaman las piezas con las que están hechos estas máquinas? ¿Por qué es más fácil utilizar una maquina o aparato para levantar o detectar plástico con ayuda de un robot, que, con las manos, así como medir la humedad y la temperatura? ¿Consideras que al utilizar un robot optimizarías los tiempos para levantar el plástico y la medición de humedad y temperatura? ¿Cómo puede un estudiante aplicar los conocimientos de la programación de microcontroladores y el uso de compuerta lógicas para ayudar a las campañas para el cuidado del medio ambiente? ¿Es posible que las carreras tecnológicas como la robótica contribuyan en actividades que promuevan la responsabilidad social sin perder su esencia emprendedora de proyectos para mejorar la calidad de vida de la humanidad? ¿De qué manera?
Producto esperado	Propuesta de prototipo robótico



Evaluación diagnóstica Submódulo I

NOMBRE: _____**GRUPO:** _____

Instrucciones: Lea las siguientes preguntas y subraye la respuesta correcta.

1. Circuito integrado capaz de ejecutar las órdenes grabadas en su memoria.

a) Pic16f86 b) Microcontrolador c) Microprocesador

2. Parte de la computadora diseñada para llevar a cabo o ejecutar los programas.

a) Microprocesador b) Microcontrolador c) CPU

3. Software que traduce un programa escrito en un lenguaje de alto nivel.

a) Simulador b) Compilador c) Ensamblador

4. Representación de una variable que puede ser cuantitativa o cualitativa que indica un valor que se le asigna a las cosas y se representa a través de una secuencia de símbolos, números o letras.

a) Instrucción b) Variable c) Dato

5. Es donde se guarda (y se recupera) datos que se utilizan en un programa.

a) Variable b) Instrucción c) Código

6. Es un evento de origen interno o externo que, si es atendido, hace que el microcontrolador interrumpa la ejecución del programa en curso y en su lugar ejecute las instrucciones de otro programa.

a) Interrupción b) Subrutina c) Variable

7. Para llevar a cabo este tipo de control o método, se debe modificar el ciclo de trabajo de la señal generada.

a) Temporizar. b) Modulación por ancho de pulso (PWM) c) Sincronizar

8. ¿Cuáles son las dos maneras en las que se lleva a cabo la comunicación binaria?

a) Paralela y síncrona. b) Síncrona y asíncrona. c) Paralela y serial.

9. ¿A qué número decimal corresponde el número binario 1110?

a) 14 b) 3 c) 2

10. Incluye en su interior las tres unidades funcionales principales de una computadora: unidad central de procesamiento (CPU), memoria y periféricos de entrada y salida.

a) Microprocesador b) Las otras dos opciones c) Microcontrolador

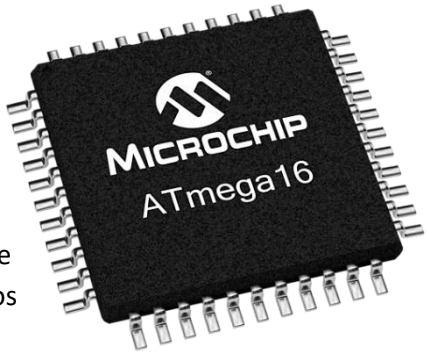
Lectura 1. "Microcontroladores y Microprocesadores"

En la actualidad es muy común que desde pequeños utilicemos dispositivos electrónicos ya sea por moda o por alguna necesidad, cuando utilizas algún dispositivo, ¿te has parado a pensar en cómo funciona? Y en ¿Cómo está constituido internamente? Muchos componentes que integran el sistema permiten que éste funcione. Estos componentes son controlados por el "cerebro" del dispositivo: el microprocesador y el microcontrolador. Aunque ambos son similares, existen algunas diferencias entre los dos.

Los microprocesadores y microcontroladores representan la innovación en el desarrollo de la tecnología electrónica en más de medio siglo, los aparatos que los utilizan han cambiado la forma de trabajar e investigar de la humanidad, en la historia ninguna herramienta creada por el hombre, tenía la capacidad de crear otras y acelerar su evolución, hoy en día muchos instrumentos, electrodomésticos y en general cualquier dispositivo electrónico utiliza alguno de estos dos componentes para optimizar su funcionamiento.

Los nuevos dispositivos están diseñados con nuevas tecnologías aplicadas tanto a la electrónica del hardware como al desarrollo del software que lo controla, se relacionan entonces varios componentes que parten del microprocesador, tal es el caso del microcontrolador. No solo encontramos microprocesadores y microcontroladores, también existen evoluciones de estos aplicados a situaciones particulares en la industria y consumo, tal es el caso de los DSP (Procesadores Digitales de Señales) y PLC (Controladores Lógicos Programables).

Las aplicaciones de control, medición, instrumentación, entretenimiento y consumo son las promotoras y fuente del creciente mercado tanto de microprocesadores como de microcontroladores. El panorama es alentador por la expectativa de nuevos productos, donde las aplicaciones están limitadas por el ingenio y la imaginación de los programadores.



Micocontroladores

Empezamos con los microcontroladores (según su nomenclatura µC o las siglas MCU). Podemos decir que un µC es un circuito integrado capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica.

Incluye en su interior las tres unidades funcionales principales de una computadora: **unidad central de procesamiento (CPU), memoria y periféricos de entrada y salida.**

A diferencia de los microprocesadores de propósito general, como los que se usan en los computadores PC, los microcontroladores son unidades autosuficientes y más económicas. El funcionamiento de los microcontroladores está determinado por el programa almacenado en su memoria. Este puede escribirse

en distintos lenguajes de programación. Además, la mayoría de los microcontroladores actuales pueden programarse repetidas veces.

Por las características mencionadas y su alta flexibilidad, los microcontroladores son ampliamente utilizados como el cerebro de una gran variedad de sistemas embebidos que controlan máquinas, componentes de sistemas complejos, como aplicaciones industriales de automatización y robótica, domótica, equipos médicos, sistemas aeroespaciales, e incluso dispositivos de la vida diaria como automóviles, hornos de microondas, teléfonos y televisores.

Características principales:

- ***Unidad de Procesamiento Central (CPU):*** Típicamente de 8 bits, pero también las hay de 4, 32 y hasta 64 bits con arquitectura Harvard, con memoria/bus de datos separada de la memoria/bus de instrucciones de programa, o arquitectura de Von Neumann, también llamada arquitectura Princeton, con memoria/bus de datos y memoria/bus de programa compartidas.
- ***Memoria de Programa:*** Puede ser una memoria ROM (Read-Only Memory), EPROM (Electrically Programmable ROM), EEPROM (Electrically Erasable/Programmable ROM) o Flash que almacena el código del programa que típicamente puede ser de 1 kilobyte a varios megabytes.
- ***Memoria de Datos:*** Es una memoria RAM (Random Access Memory) que típicamente puede ser de 1, 2 4, 8, 16, 32 kilobytes.
- ***Generador del Reloj:*** Usualmente un cristal de cuarzo de frecuencias que genera una señal oscilatoria de entre 1 a 40 MHz, o también resonadores o circuitos RC.
- ***Interfaz de Entrada/Salida:*** Puertos paralelos, seriales (UARTs, Universal Asynchronous Receiver/Transmitter), IC (Inter-Integrated Circuit), Interfaces de Periféricos Seriales (SPIs, Serial Peripheral Interfaces), Red de Área de Controladores (CAN, Controller Area Network), USB (Universal Serial Bus).

Proceso de Desarrollo

El proceso de desarrollo de una aplicación basada en microcontroladores se compone de las siguientes etapas principales, las cuales se explican en más detalle en las siguientes sub-secciones.

- ***Desarrollo de software:*** Esta etapa corresponde a la escritura y compilación/ensamblaje del programa que rige las acciones del microcontrolador y los sistemas periféricos conectados a este.
- ***Programación del microcontrolador:*** En esta etapa el código de máquina correspondiente al programa desarrollado en la etapa anterior se descarga en la memoria del microcontrolador.
- ***Prueba y verificación:*** Por último, el microcontrolador debe conectarse al circuito base y someterse a pruebas para verificar el funcionamiento correcto del programa.

Desarrollo del software

En esta etapa consiste en escribir y compilar/ensamblar el programa que determinar a las acciones del microcontrolador y su funcionamiento. Existen distintas maneras de desarrollar el programa, dependiendo del lenguaje inicial que se utiliza para escribir el programa:

El método básico es escribir el programa en lenguaje Assembler (lenguaje Ensamblador, un lenguaje de bajo nivel, muy cercano al lenguaje máquina) en un archivo de texto con extensión .asm y luego utiliza un compilador para generar un archivo en lenguaje de máquina, también denominado código de máquina o código objeto (object code), compuesto por instrucciones en código binario que son directamente entendidas por la CPU del microcontrolador.

El compilador de Assembler normalmente genera un archivo con extensión .hex (por hexadecimal), o (por objeto), .bin (por binario), o .coff (common object file format) dependiendo del programa.

Otra alternativa es emplear un lenguaje de alto nivel con una mayor cantidad de abstracciones, la cuales son más fáciles de usar y reducen los tiempos de desarrollo.

Tal vez los lenguajes de alto nivel más comunes para la programación de microcontroladores son C y C++, pero también existen otros lenguajes variantes del BASIC y el Pascal.

Una vez escrito el programa en el lenguaje de alto nivel, será necesario emplear un compilador para traducir, ya sea a lenguaje Assembler o directamente a lenguaje de máquina.

Programación del microcontrolador

Este proceso corresponde a utilizar un programa en el microcontrolador que toma el código ensamblado (.hex, .o, .bin, .coff) para el microcontrolador específico y lo envía mediante algún puerto (serial, paralelo, USB, etc.) a un dispositivo que lo escribe en la memoria del microcontrolador.

Se acostumbra denominar programador tanto al software como al hardware involucrado para este propósito, lo cual puede prestarse a confusión.

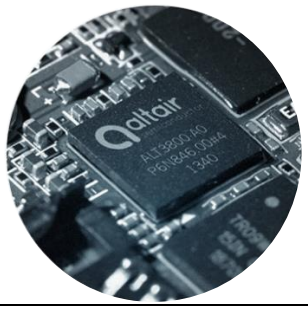
Es importante mencionar que no deben confundirse los términos desarrollo o programación del software y programación del microcontrolador, el primero se refiere a escribir el programa, mientras que el segundo se refiere transfiere el código de máquina a la memoria del microcontrolador.

Prueba y verificación

Una vez programado el microcontrolador, se puede instalar en el circuito final para comprobar su adecuado funcionamiento.

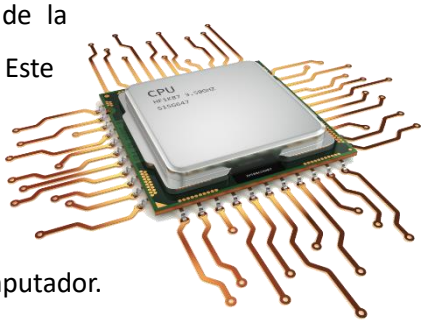
Existen herramientas de software que permiten simular el comportamiento de un µC, muy útiles cuando el programa alcanza cierta complejidad. Para resolver problemas en un circuito real, el instrumento más utilizado es el analizador lógico.

Algunas familias de Microcontroladores

FAMILIA	DESCRIPCIÓN
"ALTAIR" 	ALTAIR es el nombre genérico de una familia de microcontroladores de propósito general compatibles con la familia 51. Todos ellos son programables directamente desde un equipo PC mediante nuestro lenguaje macroensamblador, o bien mediante otros lenguajes disponibles para la familia 51 (BASIC, C, …). Los microcontroladores ALTAIR disponen de un microprocesador de 8 bits 100% compatible a nivel de código, 256 bytes de memoria interna, 128 registros especiales de función, puertos de entrada/salida de propósito general, 111 instrucciones y posibilidad de direccionar 128 Kbyte.
"INTEL" 	El 8051 es el primer microcontrolador de la familia introducida por Intel Corporation. La familia 8051 de microcontroladores son controladores de 8 bits capaces de direccionar hasta 64 Kbyte de memoria de programa y una separada memoria de datos de 64 Kbyte. El 8031(la versión sin ROM interna del 8051, siendo esta la única diferencia) tiene 128 bytes de RAM interna (el 8032 tiene RAM interna de 256 bytes y un temporizador adicional).
"SIEMENS" 	El Siemens SAB80C515 es un miembro mejorado de la familia 8051 de microcontroladores. El 80C515 es de tecnología CMOS que típicamente reduce los requerimientos de energía comparado a los dispositivos no-CMOS. Las características que tiene frente al 8051 son más puertos, un versátil convertidor análogo a digital, un optimizado Timer 2, un watchdog timer, y modos de ahorro de energía sofisticados. El 80C515 es completamente compatible con el 8051. Usa el mismo conjunto de instrucciones del lenguaje assembly MCS-51. Las nuevas facilidades del chip son controladas y monitoreadas a través de SFRs adicionales.
"MOTOROLA" 	El 68hc11 de la familia Motorola, es un potente microcontrolador de 8 bits en su bus de datos, 16 bits en su bus de direcciones, con un conjunto de instrucciones que es similar a los más antiguos miembros de la familia 68xx (6801, 6805, 6809). Dependiendo del modelo, el 68hc11 tiene internamente los siguientes dispositivos: EEPROM o OTPROM, RAM, digital I/O, timers, A/D converter, generador PWM, y canales de comunicación sincrónica y asincrónica (RS232 y SPI). La corriente típica que maneja es menor que 10ma.

"MICROCHIP"	
	Los microcontroladores PIC de Microchip Technolohy Inc. combinan una alta calidad, bajo coste y excelente rendimiento. Un gran número de estos microcontroladores son usados en una gran cantidad de aplicaciones tan comunes como periféricos del ordenador, datos de entrada automoción de datos, sistemas de seguridad y aplicaciones en el sector de telecomunicaciones. Tanto la familia del PIC16XX como la del PIC17XX están apoyadas por un rango de usuario de sistemas de desarrollo amistosos incluso programadores, emuladores y tablas de demostración. Así mismo ambas familias están apoyadas por una gran selección de software incluyendo ensambladores, linkadores, simuladores, etc.

Microprocesadores



Continuamos con los microprocesadores o µP, éste es la parte de la computadora diseñada para llevar a cabo o ejecutar los programas. Este viene siendo el cerebro de la computadora, el motor o el corazón de esta máquina. Ejecuta instrucciones que se le dan a la computadora a muy bajo nivel haciendo operaciones lógicas simples, como sumar, restar, multiplicar y dividir. El microprocesador, es el cerebro del computador. Es un chip, un tipo de componente electrónico en cuyo interior existen miles o millones de elementos llamados transistores, cuya combinación permite realizar el trabajo que tenga encomendado el chip. El microprocesador en su interior contiene la Unidad de Procesamiento Central (CPU), también llamada procesador, de un computador.

La CPU está formada por la Unidad de Control, que interpreta las instrucciones, y el camino de datos, que las ejecuta.

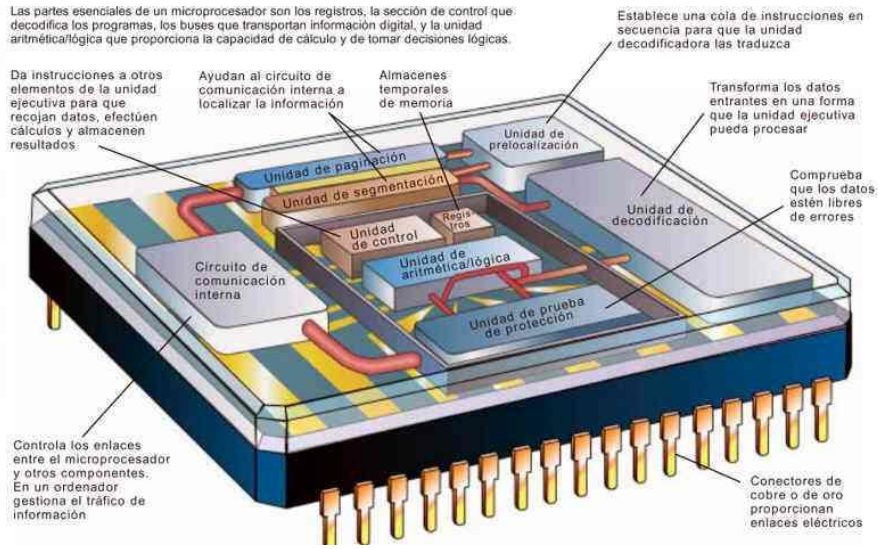
Los pines de un microprocesador sacan al exterior las líneas de sus buses de direcciones, datos y control, para permitir conectarle con la memoria y los módulos de Entrada/Salida para configurar un computador implementado por varios circuitos integrados. Se dice que un microprocesador es un sistema abierto porque su configuración es variable de acuerdo con la aplicación a la que se destine. En otras palabras, el microprocesador es como la computadora digital porque ambos realizan cálculos bajo un programa de control.

En un microprocesador se pueden diferenciar diversas partes:

- El encapsulado: es lo que rodea a la oblea de silicio en sí, para darle consistencia, impedir su deterioro y permitir el enlace con los conectores externos que lo acoplaron a su zócalo a su placa base.
- La memoria caché: es una memoria ultrarrápida que emplea el microprocesador para tener a mano ciertos datos que predeciblemente serán utilizados en las siguientes operaciones sin tener que acudir a la memoria RAM reduciendo el tiempo de espera.

- Coprocesador Matemático: Es la parte del micro especializada en esa clase de cálculos matemáticos, antiguamente estaba en el exterior del microprocesador en otro chip. Esta parte está considerada como una parte "lógica" junto con los registros, la unidad de control, memoria y bus de datos.
- Los registros: son básicamente un tipo de memoria pequeña con fines especiales que el microprocesador tiene disponible para algunos usos particulares. Hay varios grupos de registros en cada procesador. Un grupo de registros está diseñado para control del programador y hay otros que no son diseñados para ser controlados por el µP pero que la CPU los utiliza en algunas operaciones. En total son treinta y dos registros.
- La memoria: es el lugar donde el procesador encuentra sus instrucciones de programa y sus datos. Tanto los datos como las instrucciones están almacenados en memoria, y el procesador los toma de ahí. La memoria es una parte interna de la computadora y su función esencial es proporcionar un espacio de trabajo para el microprocesador.
- Puertos: es la manera en que el microprocesador se comunica con el mundo externo. Un puerto es parecido a una línea de teléfono. Cualquier parte de la computadora con la cual el procesador necesita comunicarse, tiene asignado un número de puerto que el procesador utiliza como un número de teléfono para llamar al circuito o a partes especiales.

Estructura Interna Real de un Microprocesador.



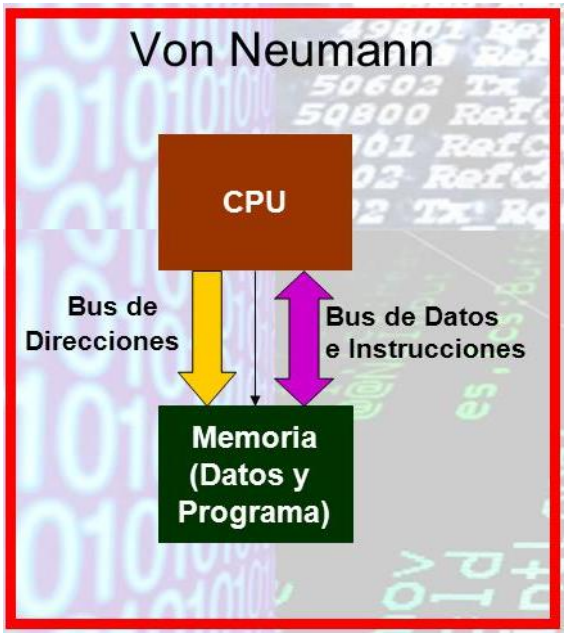
Actividad 1. Diferencias entre microcontroladores y microprocesadores

Instrucciones: Identifica y clasifica las diferencias entre los microcontroladores y microprocesadores presentes en la lectura. De igual forma realiza una investigación bibliográfica si la información necesaria no se encuentra en la lectura.

CARACTERÍSTICA	MICROPROCESADOR	MICROCONTROLADOR
CPU o procesador		
Memorias RAM y ROM		
Velocidad de Operación		
Tamaño		
Costos		
Interferencia		

Lectura 2. Arquitectura de un microcontrolador

Arquitectura Von Neumann, La arquitectura tradicional:



La arquitectura tradicional de computadoras y microcontroladores se basa en el esquema propuesto por John Von Neumann, en el cual la unidad central de proceso, o CPU, está conectada a una memoria única que contiene las instrucciones del programa y los datos. El tamaño de la unidad de datos o instrucciones está fijado por el ancho del bus de la memoria. Las dos principales limitaciones de esta arquitectura tradicional son:

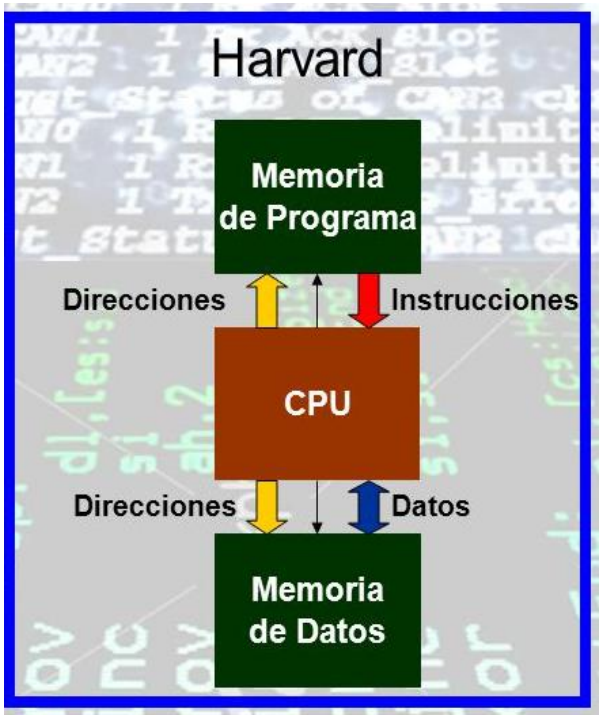
- Que la longitud de las instrucciones está limitada por la unidad de longitud de los datos, por lo tanto, el microprocesador debe hacer varios accesos a memoria para buscar instrucciones complejas.
- La velocidad de operación (o ancho de banda de operación) está limitada por el efecto de cuello de botella que significa un bus único para datos e instrucciones que impide superponer ambos tiempos de acceso.

La arquitectura Von Neumann permite el diseño de programas con código automodificable, práctica bastante usada en las antiguas computadoras que solo tenían acumulador y pocos modos de direccionamiento, pero innecesaria, en las computadoras modernas.

La arquitectura Harvard y sus ventajas:

La arquitectura conocida como Harvard, consiste simplemente en un esquema en el que el CPU está conectado a dos memorias por intermedio de dos buses separados. Una de las memorias contiene solamente las instrucciones del programa, y es llamada Memoria de Programa. La otra memoria solo almacena los datos y es llamada Memoria de Datos. Ambos buses son totalmente independientes y pueden ser de distintos anchos. Para un procesador de Set de Instrucciones Reducido, o RISC (Reduced Instrucción Set Computer), el set de instrucciones y el bus de la memoria de programa pueden diseñarse de manera tal que todas las instrucciones tengan una sola posición de memoria de programa de longitud. Además, como los buses son independientes, el CPU puede estar accediendo a los datos para completar la ejecución

de una instrucción, y al mismo tiempo estar leyendo la próxima instrucción a ejecutar. Podemos observar claramente que las principales ventajas de esta arquitectura son:



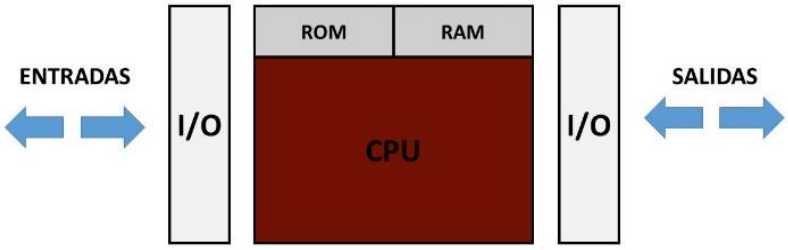
- El tamaño de las instrucciones no está relacionado con el de los datos, y por lo tanto puede ser optimizado para que cualquier instrucción ocupe una sola posición de memoria de programa, logrando así mayor velocidad y menor longitud de programa.

- El tiempo de acceso a las instrucciones puede superponerse con el de los datos, logrando una mayor velocidad de operación.

Una pequeña desventaja de los procesadores con arquitectura Harvard, es que deben poseer instrucciones especiales para acceder a tablas de valores constantes que pueda ser necesario incluir en los programas, ya que estas tablas se encontraran físicamente en la memoria de programa (por ejemplo, en la EPROM de un microprocesador).

Arquitectura interna de un microcontrolador

CPU (unidad central de proceso): Podemos decir que la CPU, siglas en inglés de unidad central de proceso, es el núcleo del microcontrolador. Se encarga de ejecutar las instrucciones almacenadas en la memoria, de la que hablaremos más adelante. Es lo que habitualmente llamamos procesador o microprocesador, término que a menudo se confunde con el de microcontrolador. En esta línea cabe aclarar que, tal y como estamos viendo, ambos términos no son lo mismo: el microprocesador es una parte de un microcontrolador y sin él no sería útil; un microcontrolador, en cambio, es un sistema completo que puede llevar a cabo de forma autónoma una labor.



Memoria: Entendemos por memoria los diferentes componentes del microcontrolador que se emplean para almacenar información durante un periodo determinado de tiempo. La información que necesitaremos durante la ejecución del programa será, por un lado, el propio código, y por otro, los diferentes datos que usemos durante la ejecución de este. Hablaremos por tanto de memoria de programa y de memoria de datos, respectivamente.

La diferente naturaleza de la información que hay que almacenar hace necesario el uso de diferentes tipos memorias. Sin hacer especial énfasis en este apartado, sí habrá que tener en cuenta una clasificación básica, que distingue entre memoria volátil y no volátil. La primera es aquella que pierde la información que almacena al desconectarla de la alimentación; la segunda, como resulta obvio, no. Por lo tanto, se hace evidente que al menos la memoria de programa deberá ser no volátil: no sería práctico que el programa grabado en el microcontrolador se borrará cada vez que apagáramos el dispositivo. Con respecto a la memoria de datos, diremos por el momento según la situación puede interesarnos una u otra.

Unidades de entrada/salida: Las unidades de entrada/salida son los sistemas que emplea el microcontrolador para comunicarse con el exterior. Imaginemos una televisión: por un lado, tiene un dispositivo de salida, como es la pantalla, y, por otro lado, de entrada, como son los botones de subir o bajar volumen y de cambio de canal. Así, los dispositivos de entrada nos permitirán introducir información en el microcontrolador y los de salida nos servirán para que éste la saque al exterior.

Arquitectura RISC y CISC

RISC (Reduced Instruction Set Computer) – Computadora con Juego de Instrucciones Reducidas.

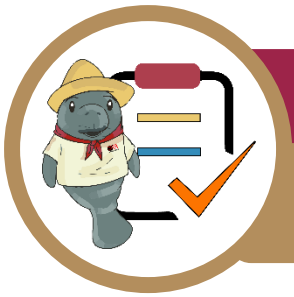
En este caso la idea es que el microcontrolador reconoce y ejecuta sólo operaciones básicas (sumar, restar, copiar etc…) Las operaciones más complicadas se realizan al combinar éstas (por ejemplo, multiplicación se lleva a cabo al realizar adición sucesiva). Es como intentar explicarle a alguien con pocas palabras cómo llegar al aeropuerto en una nueva ciudad. Sin embargo, no todo es tan oscuro. Además, el microcontrolador es muy rápido así que no es posible ver todas las "acrobacias" aritméticas que realiza. El usuario sólo puede ver el resultado final de todas las operaciones. Por último, no es tan difícil explicar dónde está el aeropuerto si se utilizan las palabras adecuadas tales como: a la derecha, a la izquierda, el kilómetro etc.

CISC (Complex Instruction Set Computer) – Computadoras con un juego de instrucciones complejo.

¡CISC es opuesto a RISC! Los microcontroladores diseñados para reconocer más de 200 instrucciones diferentes realmente pueden realizar muchas cosas a alta velocidad. No obstante, uno debe saber cómo utilizar todas las posibilidades que ofrece un lenguaje tan rico, lo que no es siempre tan fácil.

Actividad 2. Compiladores para microcontroladores

Instrucciones: Integrados en equipos de 5 estudiantes realiza una revisión bibliográfica de los compiladores para microcontroladores para preparar una exposición, debes preparar tu material de apoyo físico o digital para presentarlo en plenaria y socializar la información.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 1

Actividad 2: Compiladores para microcontroladores

DATOS GENERALES						
Nombre(s) del alumno (s)				**Matriculas(s)**		
Producto: Exposición				**Fecha**		
Formación Laboral Básica: Robótica **Submódulo 1:** Microcontroladores y microprocesadores				**Periodo: 2026 – 2027ª**		
Nombre del docente:				**Firma del docente**		
No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	El equipo completo demuestra preparación al realizar la exposición.			1		
2.	El equipo está organizado al momento de exponer.			1		
3.	El material de apoyo es visualmente atractivo y sin faltas de ortografía.			1		
4.	Manejan la exposición suscitando la participación de la audiencia.			1		
5.	Usan un tono de voz adecuado.			1		
6.	Mantienen el control del grupo.			1		
7.	Desarrollan de manera óptima una dinámica, dando instrucciones claras y manteniendo el orden.			1		
8.	Resuelven dudas en caso de existir.			1		
9.	Entregó en tiempo y forma.			1		
10.	Realiza las conclusiones solicitadas.			1		
				CALIFICACIÓN		

Lectura 3. ¿Qué es Arduino?

Arduino es una placa de circuito impreso con la que, junto con unos componentes electrónicos, un microcontrolador y una serie de pines de entrada y salida, podemos crear proyectos basados en sistemas electrónicos; esto incluye materias como la robótica, la domótica u otros proyectos de carácter electrónico en los que podamos pensar.

Mediante un lenguaje de alto nivel programaremos el microcontrolador para que ejecute las acciones que deseamos llevar a cabo en el proyecto en cuestión.

Una de sus principales ventajas es que tanto los diseños como los esquemas de sus componentes son de acceso público, lo que permite a cualquier persona modificar, replicar o incluso comercializar placas basadas en Arduino, siempre que se respeten los términos de su licencia GPL. En particular, la placa Arduino UNO R3 incorpora un microcontrolador Atmel Atmega328P, caracterizado por su bajo costo y facilidad de programación, lo que lo convierte en una opción accesible para múltiples aplicaciones.

Ahora mismo, podemos encontrar en el mercado versiones más potentes de la familia Arduino, así como competidores en este terreno como las placas BeagleBone o Raspberry Pi, por ejemplo, aunque estas placas se consideran acertadamente ordenadores del tamaño de una tarjeta de crédito.

En esta guía nos centraremos en la utilización de la placa Arduino UNO R3, por lo que a continuación se muestra un esquema de sus elementos, así como su descripción.

- ***Botón* de reset.** Permite realizar un reinicio a la placa. Una vez reseteada la placa, ésta vuelve a ejecutar el programa que tiene cargado.

- ***Conector USB***. Se emplea para comunicar la placa Arduino con el PC mediante un cable USB tipo B-USB tipo A. También se utiliza para comunicamos con Arduino a través del monitor serie.
- ***Conexión 7v - 12v***. Mediante un Jack de 2.1 mm alimentaremos a la placa Arduino con un rango de tensión comprendido entre los 7 y los 12 voltios. Una tensión que suele ir bien es la que proporciona una pila de 9 voltios.
- ***Pines de alimentación***. Hay seis pines que se detallan a continuación.
  - ***IOREF***. Es un pin de referencia del voltaje al que tendrá que trabajar el microcontrolador. También se utiliza cuando conectamos *shields (placas de expansión)* a Arduino, regulando la tensión para un adecuado funcionamiento.
  - ***RESET***. Este pin tiene la misma función que el botón reset. Aquí lo encontramos en formato de pin para poder resetear la placa mediante un pulsador externo.
  - ***3,3v***. Este pin proporciona 3,3 voltios. Es posible que algún sensor o componente requiera de esa tensión para poder funcionar correctamente.
  - ***5v***. Este pin proporciona 5 voltios para alimentar los dispositivos, sensores y/o componentes electrónicos conectados a Arduino.
  - ***GND***. Aquí se conectarán los terminales de masa de los componentes electrónicos que se hayan podido conectar al terminal de 5 o 3,3 voltios de Arduino.
  - ***Vin***. Este terminal permite alimentar a Arduino de la misma forma en que se realiza en el caso del conector de alimentación mediante un conector Jack de 9 mm.
- ***Pines analógicos***. Son terminales que se emplean para comunicar la placa Arduino con el exterior, conectando sensores que les proporcionan información analógica. Se podrán configurar como entrada o salida. Arduino UNO dispone de seis pines de este tipo.
- ***Microcontrolador Atmega 328P***. Este circuito integrado es el cerebro de la placa. Es el encargado de ejecutar las instrucciones de los programas creados por el usuario.
- ***Indicador TX-RX***. Indica que Arduino se está comunicando vía serie con el PC. Cuando esto ocurre, los indicadores parpadean, alertando de la transmisión y recepción de la información transmitida entre Arduino y el ordenador.
- ***Conectores ICSP***. Se utilizan cuando se desea programar Arduino desde un entorno diferente del IDE y de la conexión típica por USB. Para realizar esta operación se requiere de un programador externo que irá conectado a los conectores mencionados. Si se desea programar Arduino de este modo, se deberá hacer en lenguaje Ensamblador o en lenguaje de alto nivel C.
- ***Indicador de encendido***. Mediante una lucecita verde indica que Arduino está alimentado correctamente y listo para programar.
- ***Indicador de carga***. Este indicador parpadea cuando se carga un programa a Arduino.
- ***Pines digitales***. Son terminales que se emplean para comunicar la placa Arduino con el exterior, conectando sensores que proporcionan información digital (5v o 0v, 1 o 0). Se podrán configurar como entrada o salida. Arduino UNO dispone de catorce pines de este tipo. Dentro de este conjunto de pines, encontramos uno que responde al nombre de A REF.

- ***A REF***. Proporciona el voltaje de referencia para los pines analógicos. Generalmente, esta referencia es de 0 a 5 voltios, pero podemos encontrarnos con componentes para Arduino que funcionan con otro rango de voltajes, por lo que los voltajes de referencia deberían ser ajustados.

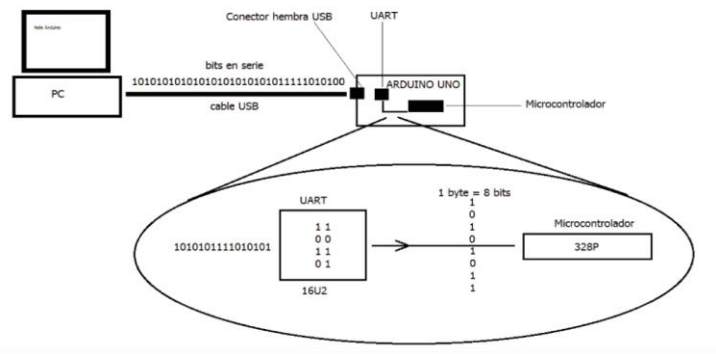
Podemos ver que lo más interesante a la hora de programar Arduino son las entradas y salidas, tanto analógicas como digitales, por donde introduciremos datos para ser procesados y así obtener un resultado que se manifestará por las salidas antes mencionadas.

Comunicación Arduino-PC

El microcontrolador 328P es un dispositivo de 8 bits, por lo que procesa instrucciones o datos de 8 en 8 bits o, lo que es lo mismo, 1 byte.

La comunicación entre el PC y Arduino se realiza por un cable o interfaz USB (Universal Serial Bus), que es una forma de transmitir los bits de uno en uno.

El problema lo podemos advertir cuando nos damos cuenta de que la transmisión de datos es en serie (un bit tras otro) y el microcontrolador necesita «paquetes» de 8 bits para procesar. La placa Arduino posee un circuito integrado adicional soldado, que une el conector hembra USB con el microcontrolador. A este circuito integrado se le denomina UART (Transmisor - Receptor Asíncrono Universal), y es el encargado de gestionar los bits y adecuarlos según se necesiten en serie o en grupos de 8 bits.



Esquema de comunicación Arduino-PC

Arduino IDE

Medio Integrado de Desarrollo de Arduino

Para programar Arduino vamos a necesitar un software que compile el programa creado y, a través de un cable USB, cargue o transfiera ese programa hasta el microcontrolador de la placa Arduino. A este software se le conoce como medio integrado de desarrollo (del inglés *Integrated Development Environment*). Tiene versiones para Windows, Linux y Mac.

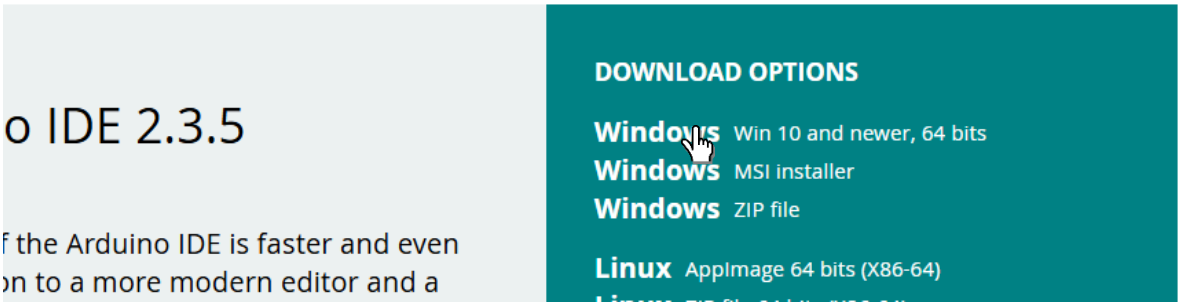
Actividad 3. Instalación de Arduino IDE

Instrucciones: Completa cada uno de los siguientes puntos hasta instalar Arduino IDE en tu computadora, posteriormente muestra el software ya instalado al docente.

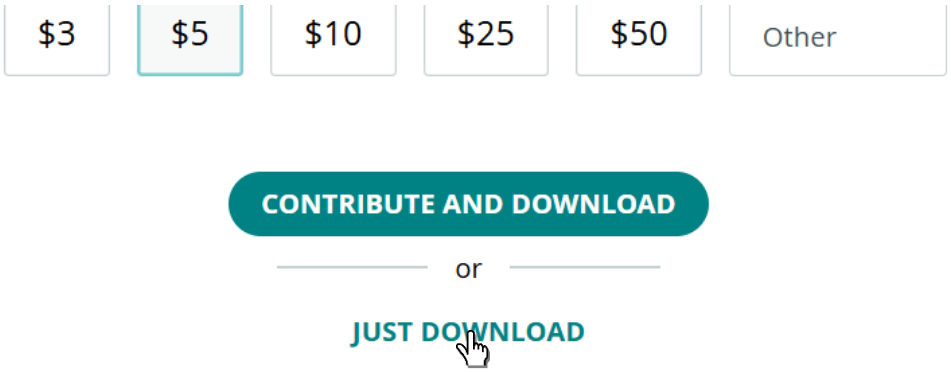
Fase 1: Instalación de Arduino IDE

1. Entra al siguiente enlace que corresponde a la página oficial de Arduino: https://www.arduino.cc/en/software

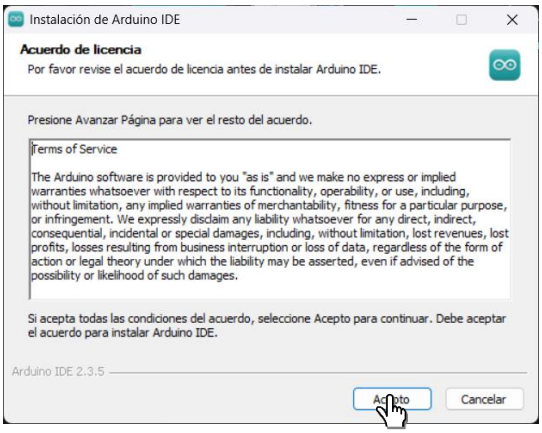
2. En el apartado de opciones de descargas (downloads options) selecciona el que corresponde a tu sistema operativo y el formato en que deseas descargar el instalador. Si tienes sistema operativo Windows se sugiere descargar la primera opción.

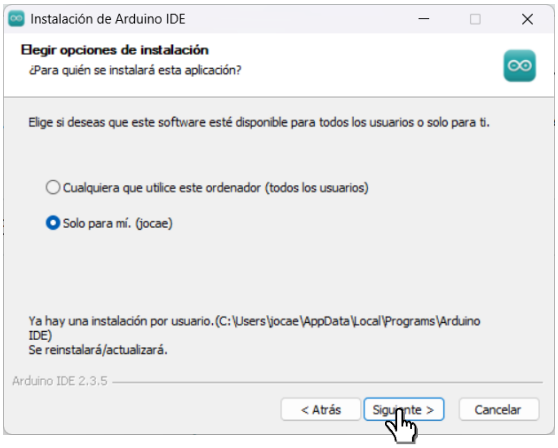


3. Aparecerá una página para hacer donaciones a los desarrolladores de Arduino, si quieres donar selecciona la cantidad, pero si no solo selecciona "JUST DOWNLOAD". Selecciona donde quieres guardar el instalador y da clic en guardar.

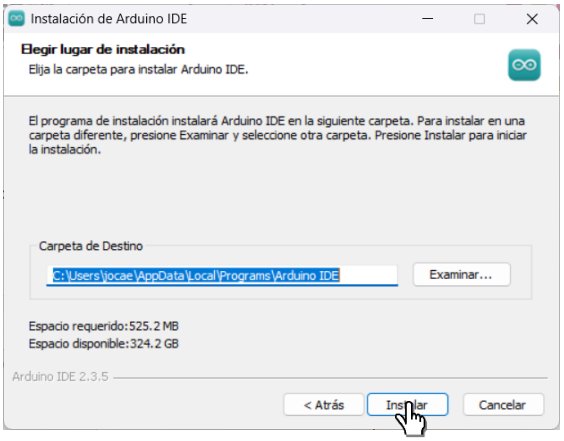


4. Ve a la carpeta donde descargaste el instalador, ubícalo y da doble clic sobre él. En la ventana que aparece acepta el acuerdo de licencia.





5. Selecciona si quieres instalar Arduino IDE para todos los usuarios de la PC o solo para ti y da clic en "Siguiente".



6. El siguiente paso es elegir la ubicación donde se instarán los archivos necesarios del Arduino IDE, comúnmente se deja la dirección que viene por default, dar clic en Instalar y esperar que se instale el programa. Da clic en "Instalar".

7. Una vez transcurrido el tiempo necesario para la instalación, aparecerá una ventana de confirmación y debes presionar "Terminar" para cerrar el asistente. Deja marcada la casilla "Ejecutar Arduino IDE".

Lectura 4. Estructura básica de un programa

Un programa puede considerarse como una secuencia de acciones (instrucciones) que manipulan un conjunto de objetos (datos).

Bloques de un programa

- ***Bloque de declaraciones:*** en él se especifican todos los objetos que utiliza el programa (constantes, variables, tablas, registros, archivos, etc.).
- ***Bloque de instrucciones*****:** constituido por el conjunto de operaciones que se han de realizar para la obtención de los resultados deseados

Partes principales de un programa

Dentro del bloque de instrucciones de un programa se pueden diferenciar tres partes fundamentales. En algunos casos, estas tres partes están perfectamente delimitadas, pero en la mayoría sus instrucciones quedan entremezcladas a lo largo del programa, si bien mantienen una cierta localización geométrica impuesta por la propia naturaleza de estas.

- ***Entrada de datos:*** la constituyen todas aquellas instrucciones que toman datos de un dispositivo externo, almacenándolos en la memoria central para que puedan ser procesados.
- ***Proceso o algoritmo:*** está formado por las instrucciones que modifican los objetos a partir de su estado inicial hasta el estado final, dejando éstos disponibles en la memoria central.
- ***Salida de resultados:*** conjunto de instrucciones que toman los datos finales de la memoria central y los envían a los dispositivos externos.

En esta guía se trabajará con el IDE de Arduino para programar microcontroladores, por esta razón a partir de ahora nos centraremos en la sintaxis, estructura y programación en lenguaje C y C++ que es el que utiliza esta plataforma.

Estructura básica de un sketch

Se denomina sketch a un programa para Arduino, y este posee la extensión .ino. Un proyecto en esta plataforma puede estar compuesto por varios archivos, con la única condición de que todos deben estar alojados en la misma carpeta donde está el archivo principal.

La estructura básica de un sketch de Arduino es bastante sencilla y, generalmente, encontraremos dos partes obligatorias, las que se encargarán de encerrar las declaraciones, estamentos y también instrucciones: **setup()** y **loop()**.

Su estructura básica es la siguiente:

```
//Primera parte
void setup() {
Instrucciones;
}
    //Segunda parte
void loop() {
Instrucciones;
}
```

La primera parte es donde se va a declarar y a introducir los datos al iniciar el programa y solo se ejecutarán una vez cuando se enciende la placa Arduino. La segunda parte es donde se vana a introducir las instrucciones que se van a repetir hasta que nosotros creamos oportuno.

Además de estas dos secciones básicas, podemos encontrar otros apartados en un sketch:

- ***Comentarios:*** se trata de indicaciones o información sobre las tareas relacionadas con una línea de código o con una parte del programa. Si se trata de un comentario de varias líneas deberá delimitarse por /* al comienzo y por */ al final. Si es un comentario de una sola línea solo se debe comenzar con //.
- ***Variables:*** son adecuadas para guardar información, pueden estar referidas a diferentes ámbitos y relacionarse con distintos tipos de datos. Las variables globales se declaran al inicio de un sketch de Arduino.
- ***Funciones:*** si tenemos bloques de código que utilizaremos a menudo, es posible crear una función para facilitar nuestro trabajo. En este sentido, una función es un bloque de código que encontramos fuera de setup() y loop(), que posee un nombre característico con uno o más parámetros de entrada, y puede devolver o no un valor.
  Para declarar una función es necesario saber el tipo de datos que esta devolverá (void, int, float, Boole, etc.), luego debemos especificar el nombre de la función y posteriormente las instrucciones que ejecutará.
  Para crear una función en un sketch de Arduino escribiremos:

  void *nombre_de_la_función* () { //void cuando no se produce ningún resultado
  instrucciones que deseamos que realice la función;
  }

  Para invocar la función simplemente escribiremos, allí donde deseemos que se ejecute ese bloque de código: ***Nombre_función* ();**

Veamos un ejemplo:

```
void loop() {
dist=0; //variable que almacena distancia de objeto en línea recta
distbarder=0; //variable que almacena distancia de la derecha
distbarizq=0; //variable que almacena distancia de la izquierda

adelante(); //invocamos a la función adelante
barrido_central(); //invocamos a la función barrido central

if (dist>25) {  //si la distancia es mayor que 25 cm…
        adelante(); //invocamos a la función adelante
}
```

En este fragmento podemos ver las invocaciones a las diferentes funciones creadas por el programador. Estas funciones son: *adelante(); y barrido_central();.*
Hay que recordar que la invocación de una función se puede realizar dentro del void loop() o dentro del void setup().

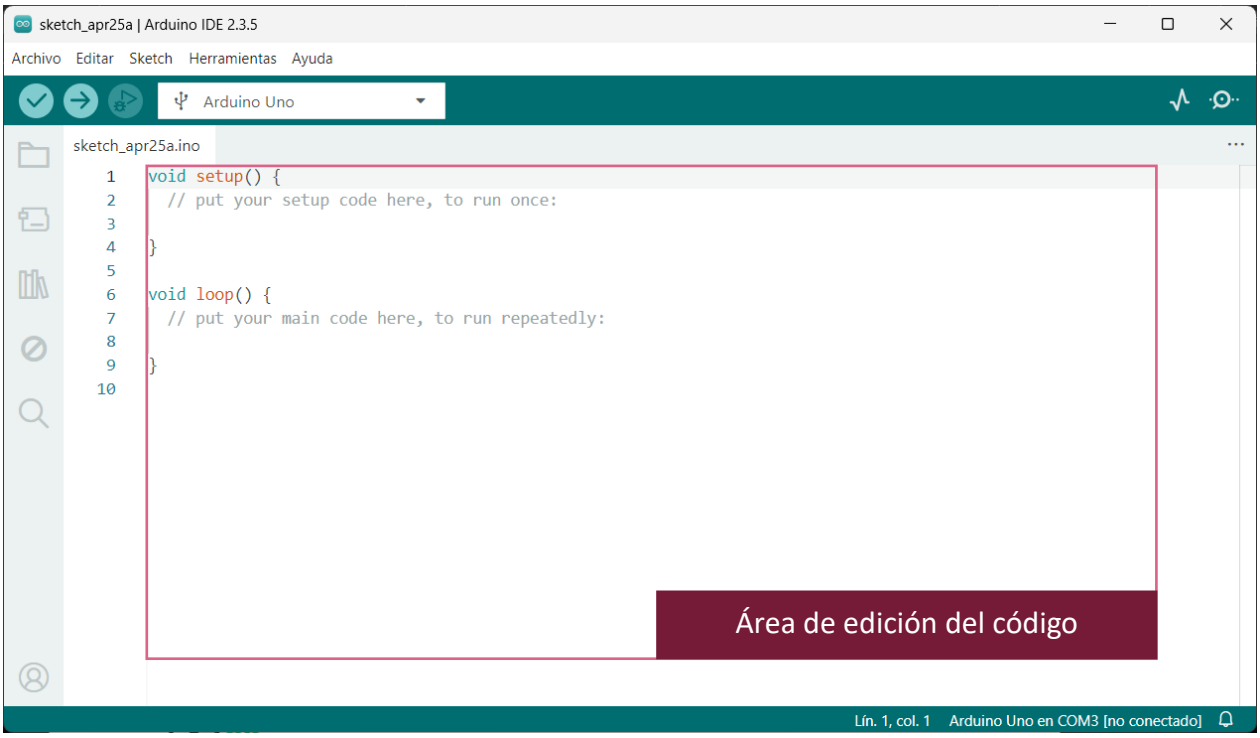
- ***Case sensitive:*** al programar en Arduino debemos tener en cuenta que es *case-sensitive,* es decir que es diferente escribir una letra en mayúscula que en minúscula, por ejemplo Arduino y Arduino serían tratados de forma distinta.
- ***Tabulaciones:*** el uso de tabulaciones no es necesario para que la compilación del sketch se realice en forma correcta, aunque se trata de una manera de escribir un código ordenado, claro y cómodo tanto para el programador como para quien debe leer o analizar el código en forma posterior.
- ***Puntos y comas:*** en un sketch de Arduino, las instrucciones deben terminar con un punto y coma, esto indica el final de cada instrucción y debe escribirse al final de cada línea, si no se escriben nos encontraremos con errores al momento de compilar.

```
3
4   volatile int D_mix;
5   volatile int D_mid;
6   volatile int D_max;
7   volatile int Front_Distance;
8   volatile int Left_Distance;
9   volatile int Right_Distance;
10  volatile int Right_IR_Value;
11  volatile int Left_IR_Value;
12
13  void Ultrasonic_obstacle_avoidance() {
14    if (Front_Distance <= D_mid) {
15      digitalWrite(2,LOW);
16      analogWrite(5,0);
17      digitalWrite(4,HIGH);
18      analogWrite(6,0);
19      if (Front_Distance <= D_mix || Left_IR_Value == 0 && Right_IR_Value == 0) {
20        digitalWrite(2,LOW);
21        analogWrite(5,(4.5 * 22.5));
22        digitalWrite(4,HIGH);
23        analogWrite(6,(4.5 * 22.5));
24        delay(300);
25        digitalWrite(2,LOW);
26        analogWrite(5,0);
27        digitalWrite(4,HIGH);
28        analogWrite(6,0);
29
```

Fragmento de un sketch en Arduino. Observa la utilización de tabulaciones, variables, funciones, etc.

Lectura 5. Edición, compilación y guardado de un sketch en Arduino

Cuando abres el Arduino IDE por primera vez o creas un nuevo sketch, verás una página como esta, donde el Arduino IDE crea un nuevo archivo para ti, llamado «sketch».



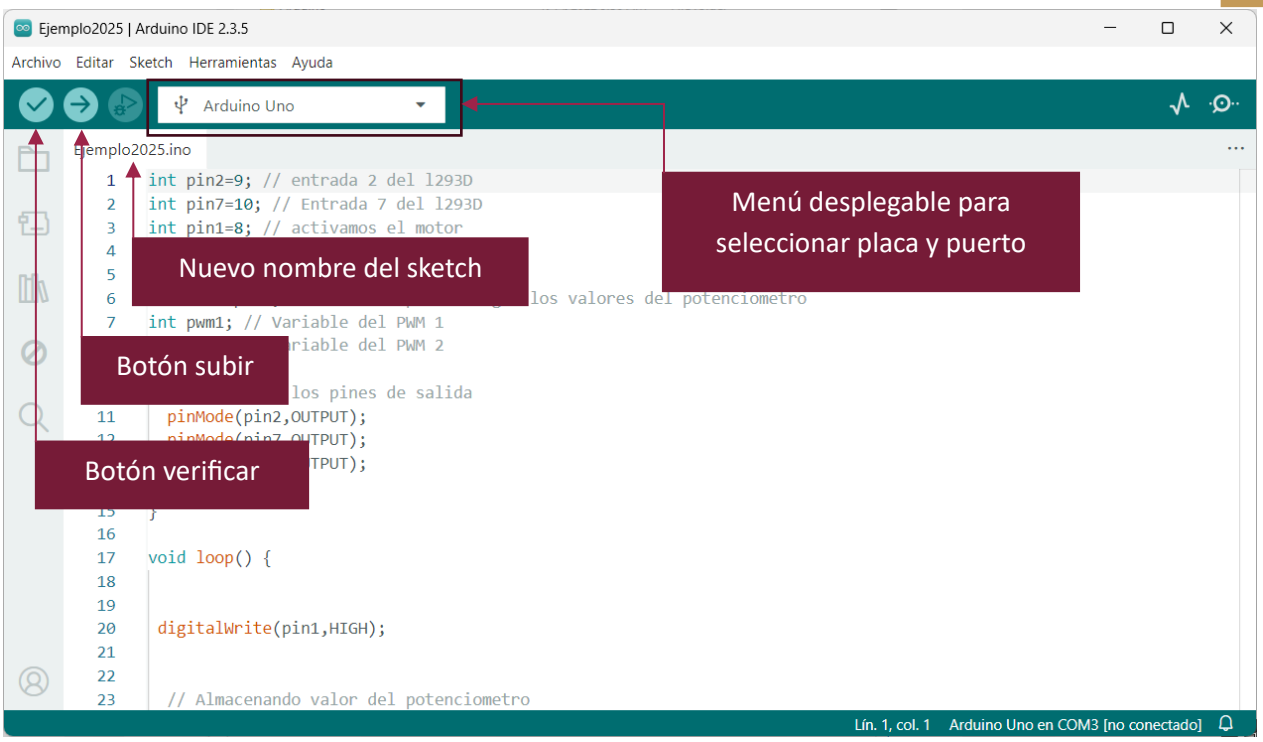
Estos archivos de sketch tienen un nombre temporal regular, del cual puedes deducir la fecha en que se creó el archivo. sketch_apr25a.ino significa primer sketch del 25 de abril (april), .ino es el formato de archivo de este sketch.

En el área del void setup() y void loop() puedes escribir las líneas de código correspondientes a tu proyecto. Si quieres crear un nuevo sketch, ve al menú **Archivo>New Sketch** o presiona las teclas **Ctrl+N.**

Una vez que hayas terminado de escribir/editar el código de tu proyecto, ve al menú **Archivo>Guardar como…** (o presiona las teclas **Ctrl+Mayús+SNN**) para guardar tu proyecto. El sketch se guarda en: *C:\Users\{tu_usuario}\Documents\Arduino* por defecto, puedes renombrarlo o elegir una nueva ruta para guardarlo.

Después de guardar exitosamente, verás que el nombre en el Arduino IDE se ha actualizado.

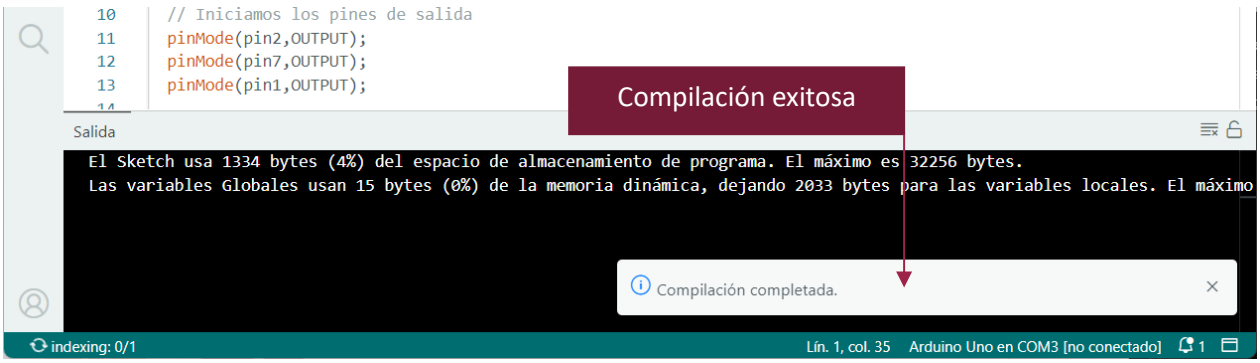
En este caso se guardó con el nombre de "Ejemplo2025", observa en la siguiente imagen.



Las placas de desarrollo de Arduino generalmente vienen con un cable USB, es momento de usarlo para conectar la placa a tu computadora.

Antes de hacer nada, debemos elegir la placa a la que vamos a subir los datos. Junto al botón "Verificar y subir", verá un menú desplegable que, en la mayoría de los casos, mostrará las placas Arduino conectadas a su ordenador. Si su placa no se detecta automáticamente, puede pulsar "Seleccionar otra placa y puerto..." en el menú desplegable y seguir las instrucciones, o ir a **Herramientas > Placa** y **Herramientas>puerto** en la barra de herramientas para seleccionar la placa y el puerto manualmente. Sabrá que hay una conexión con la placa cuando su nombre aparezca en negrita.

Haga clic en la herramienta de verificación (marca de verificación). Después de unos segundos, veremos el resultado de la acción en la consola (Compilación completada).



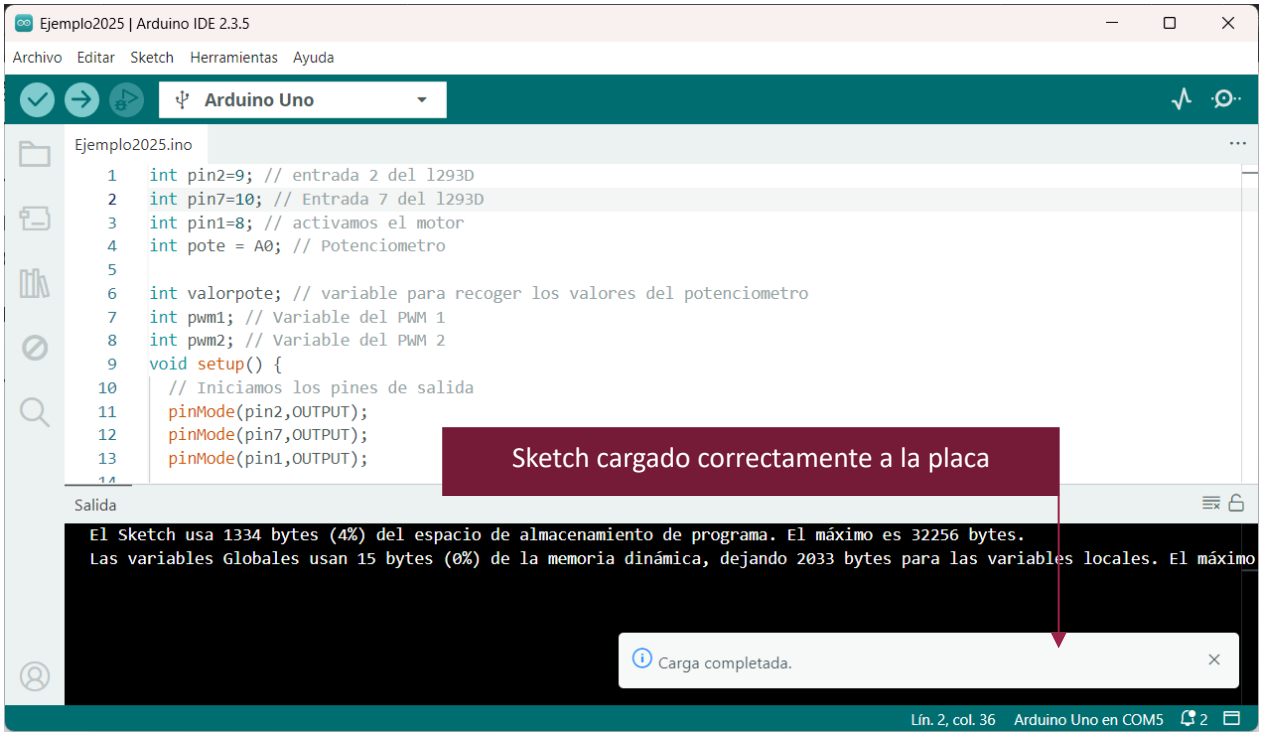
Herramienta verificar y subir

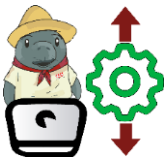
Hay dos herramientas principales para subir un sketch a una placa: **verificar y subir**. La herramienta verificar simplemente revisa el sketch, busca errores y lo compila. La herramienta subir hace lo mismo, pero al terminar de compilar el código, también lo sube a la placa.

Una buena práctica es usar la herramienta de verificación antes de intentar subir algo. Es una forma rápida de detectar errores en el código y corregirlos antes de subirlo.

Finalmente haz clic en el botón subir para que el sketch comience a cargarse en la placa.

Cuando el sketch termine de subirse, aparecerá una notificación en la esquina inferior derecha de la ventana del IDE. Por supuesto, a veces surgen complicaciones al cargar, y estos errores también se indicarán aquí.





Práctica 1. Intermitencia de dos leds

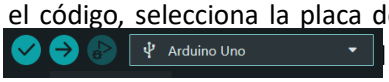
Instrucciones: Completa cada uno de los siguientes puntos para el desarrollo de las tres fases que comprenden la práctica de la intermitencia dos leds.

Fase 1. Código del circuito

En esta fase se pretende crear un programa donde existan dos diodos led para que se enciendan y se apaguen de forma síncrona, es decir, cuando un led esté encendido, el otro deberá estar apagado y viceversa. La duración de la posición de encendido y apagado deberá ser de un segundo.

1. Abre el IDE de Arduino y en un nuevo sketch copia el siguiente código, analiza bien los comentarios para que logres entender su estructura.

```
/* Código para la práctica Intermitencia de dos leds*/

int pin_rojo_1=13;    //asignamos el pin digital 13 al led rojo 1
int pin_rojo_2=12;  //asignamos el pin digital 12 al led rojo 2

void setup() {

      pinMode (pin_rojo_1, OUTPUT); //declaramos que el pin 13 será de salida
      pinMode (pin_rojo_2, OUTPUT); //declaramos que el pin 12 será de salida
      digitalWrite (pin_rojo_1, LOW); //nos aseguramos deque el led rojo 1 empezará apagado
      digitalWrite (pin_rojo_2, LOW); //nos aseguramos de que el led rojo 2 empezará apagado
}

void loop() {

      digitalWrite (pin_rojo_1, HIGH); //empezamos la secuencia de
intermitencia activando el led 1
      delay(1000) ; //esperamos 1 segundo con el led 1 activado
      digitalWrite(pin_rojo_1, LOW); //desactivamos el led rojo1
      digitalWrite(pin_rojo_2, HIGH); //activamos elledrojo2
      delay(1000) ; //esperamos 1 segundo con el led 2 activado
      digitalWrite(pin_rojo_2, LOW); //desactivamos el led rojo2
      delay(1000) ; // esperamos 1 segundo con el led rojo 2 desactivado
//el proceso vuelve a comenzar
}
```

2. Una vez copiado el código, selecciona la placa de Arduino uno en la barra superior del IDE de Arduino 

programa en el botón verificar que es una palomita ubicada también la parte superior del IDE. Si no hay errores aparece el mensaje de "Compilación completada", si los hay, aparecerán en la parte inferior en el apartado de salida.

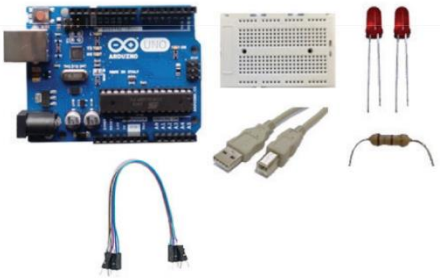
Fase 2. Cargar programa a la placa de Arduino Uno

1. Conecta la placa de Arduino Uno al PC a través del cable USB, y en la barra superior donde seleccionamos la placa de Arduino en la fase 1, selecciona el puerto com que corresponde a la placa.

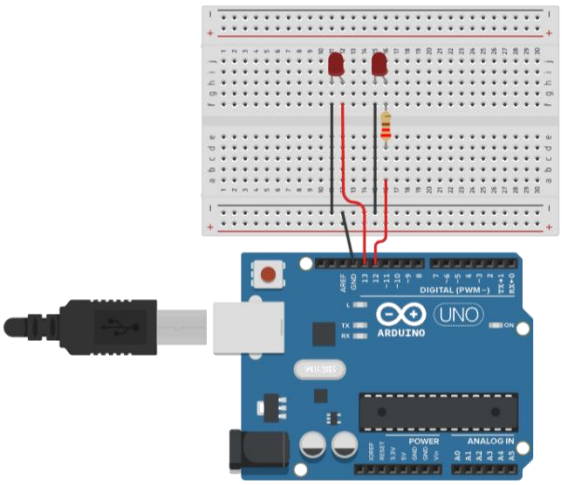
2. Hacer clic en el botón  ra cargar el código a la placa de Arduino Uno.

Fase 3. Conexiones del circuito.

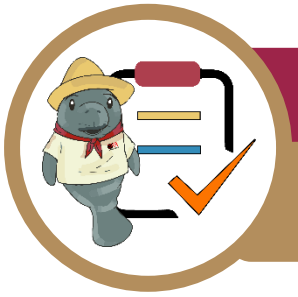
1. Materiales.
   - Placa Arduino UNO
   - Protoboard
   - Cable USB
   - Dos leds
   - Una resistencia de 220 Ohms
   - Cable conexión



2. Esquema de conexión.



3. Conecta la alimentación a la placa y comprueba del correcto funcionamiento del circuito.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 2

Práctica 1: Intermitencia de dos leds

DATOS GENERALES		
Nombre(s) del alumno (s)		**Matriculas(s)**
Producto: Reporte de práctica		**Fecha**
Formación laboral básica: Robótica **Submódulo 1:** Microcontroladores y microprocesadores		**Periodo: 2026 – 2027A**
Nombre del docente:		**Firma del docente**

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Copió y comprendió correctamente el código en el IDE de Arduino.			2		
2.	El código fue compilado sin errores (mensaje: "Compilación completada").			2		
3.	Cargó exitosamente el programa a la placa Arduino Uno.			1		
4.	Identificó y utilizó correctamente los materiales requeridos.			1		
5	Realizó correctamente el armado físico del circuito en el protoboard.			2		
6	El circuito funciona correctamente: los LEDs se encienden alternadamente.			2		
				CALIFICACIÓN		

Lectura 6. Tipos de datos y variables

Variables

Una **variable** es un pequeño contenedor de memoria que se emplean para almacenar datos, ya sean letras (alfabéticos), números (numéricos) o una combinación de ambos (alfanuméricos).

Como su nombre los indica, una variable puede alterar su contenido a loa largo del tiempo; esto es, el propio programador puede cambiar su valor de forma directa durante la escritura del programa o de forma indirecta mientras se ejecuta el programa.

Para poder utilizar las variables dentro de un programa, primero se deberán "declarar" para que el procesador las tenga en cuenta, siendo muy práctico asignar a las variables nombres que indiquen que valores van a almacenar.

Podemos diferenciar diversos tipos de variables, dependiendo de los datos que van a almacenar; entre ellos encontramos los siguientes:

Tipo de variable	Características
char	Almacena caracteres, ocupa un byte.
byte	Almacena un número entre 0 y 255.
int	Almacena un número entre -32 768 y 32 767, ocupa 2 bytes.
unsignedint	Almacena un número entre 0 y 65 535, ocupa 2 bytes.
long	Ocupa 4 bytes, almacena un número entre -2 147 483 648 y 2 147 483 647.
float	Almacena números decimales que van entre -3.4028235E+38 y +3.4028235E+38.
double	Almacena números decimales, pero dispone de 8 bytes.
string	Almacena cadenas de texto.

Es así como, para declarar una variable, debemos asignarle un nombre, así como el tipo de dato que almacenará; además es posible asignarle un valor de forma inmediata, por ejemplo:

```
char CaracterDos='b';
byte Numero=192;
int MiEntero;
unsignedint numPos=2212;
```

Las variables deben tomar nombres más descriptivos para que podamos hacer el código más legible. Como vemos, hemos creado cuatro variables, aunque solo en tres de ellas asignamos un valor, dejamos la variable de tipo int sin inicializar. Veamos un ejemplo de código donde se utilizan variables de tipo int:

```
int f = 7;
int g = 9;
int h = f + g;

void setup ( )
{
  Serial.begin(9600);
  Serial.print("f + g equals " ) ;
  Serial.println(String(h));
}

void loop ()
{
}
```

En este código se han creado las variables f, g y h, en las dos primeras se almacenan los datos 7 y 9, respectivamente, mientras que en la variable h almacenamos el resultado de sumar las variables f y g.

Las variables pueden ser globales o locales, la diferencia entre ellas radica en el alcance que poseen, es decir, desde qué lugares del sketch pueden ser utilizadas. Una variable es global cuando podemos acceder a ella desde cualquier lugar del sketch, es decir, está disponible tanto para **setup()** como para **loop()**. Veamos un ejemplo:

```
int a= 2 ;

void setup()
{
 Serial.begin(9600);
 Serial.println(String(a));
}

void loop()
{
 a = a + 2;
 Serial.println(String(a));
 delay(3000);
}
```

Constantes

Podemos entender a las constantes como variables de solo lectura, es decir, no es posible modificar el valor que les fue asignado; si intentamos hacerlo, el compilador nos mostrará un error. La definición de una constante se hace de forma similar a una variable, pero debemos preceder su declaración con la palabra **const**. Por ejemplo, **const int numero1=150**. En este caso, se trata de una constante de nombre **numero1**, con el valor **150**.

En este caso hemos declarado la variable fuera de **setup()** y **loop()**, por lo tanto, su ámbito es global y la podemos utilizar en cualquier parte del sketch.

Por otra parte, una variable es local cuando se declara dentro de un ámbito específico (dentro de corchetes), de esta forma no podrá ser usada en un ámbito distinto. Veamos un ejemplo basado en el código anterior:

```
void setup ()
{
int a= 2 ;
 Serial.begin(9600);
 Serial.println(String(a));
}

void loop ()
{
 a = a + 2;
 Serial.println(String(a));
 delay(3000);
}
```

En este caso, hemos declarado la función dentro de **setup()**, por lo tanto, podemos utilizarla solo dentro de ese ámbito. Al llamarla desde **loop()** obtendremos un error de compilación. Una posible solución sería la siguiente, aunque no es un código correcto pues debemos tener en cuenta que la variable **int** dentro de **loop()** se creará y se destruirá con cada iteración, pero no presentará errores de compilación:

```
void setup ()
{
int a= 2;
  Serial.begin(9600);
  Serial.println(String(a));
}

void loop ()
{
int a= 2 ;
  a = a + 2 ;
  Serial.println(String(a));
  delay(3000);
}
```

Consejos para declarar variables

En primer lugar, necesitamos examinar muy bien el ámbito que deseamos asignarle a la nueva variable; dependiendo de esto, la escribiremos en un lugar o en otro. Por otra parte, también es relevante darle un nombre descriptivo, de esa forma será más fácil reconocerla en códigos extensos, por ejemplo, es mejor una variable denominada **SensorDeDistancia** que una llamada **VariableDos**.

Datos y operadores

Dos conceptos relevantes que debemos tener en cuenta a la hora de programar para Arduino son los datos y los operadores. Los datos son importantes pues se trata de la materia prima con la que trabajaremos y,

por otra parte, los operadores nos permitirán generar ciertas acciones sobre los datos, de esta forma obtendremos resultados.

En primer lugar, analizaremos los diferentes tipos de datos con los que podemos trabajar, esto se encuentra íntimamente relacionado con las variables que se mencionaron anteriormente, pues, para declarar una variable, definimos el tipo de dato que almacenará. Los tipos de datos más comunes utilizados en un ambiente Arduino se muestran en la siguiente tabla.

Tipo de dato	Tamaño	Descripción
int	16 bits	Número con signo entre -32 768 y 32 767 sin decimales. Se trata del tipo de dato más usado para propósitos generales en Arduino.
long	32 bits	Número con signo entre -2 147 483 648 y 2 147 483 647 sin decimales.
float	32 bits	Número con signo entre -3.4028235E38 y 3.4028235E38 con decimales.
byte	8 bits	Número sin signo entre 0 y 255 sin decimales.
word	16 bits	Número sin signo entre 0 y 65 535 sin decimales.
char	8 bits	Número con signo entre -128 y 127.
boolean	8 bits	Lógico simple, verdadero/falso.
unsigned int	16 bits	Número sin signo entre 0 y 65 535 sin decimales.
unsigned long	32 bits	Número sin signo entre 0 y 4 294 967 295 sin decimales. Se utiliza para guardar los resultados de la función millis().
unsigned char	8 bits	Número sin signo entre 0 y 255.
array	-	Colección de valores. Pueden existir arrays de todos los tipos de datos vistos en este apartado, pero todos los elementos del array tienen que ser del mismo tipo de datos.
String	-	Cadena de caracteres.
void	-	Únicamente es utilizado para indicar que una función o método no devuelve nada.

El tamaño necesario para cada tipo de dato es un aspecto relevante a la hora de escribir código. Debemos recordar que, por ejemplo, el procesador ATmega328P es nativo de 8 bits, por lo que posee soporte integrado para números de punto flotante. Para trabajar con tamaños de memoria mayores a 8 bits, el procesador gestionará secuencias de código que le permitan segmentar el trabajo, para almacenar el resultado donde corresponda.

De esta forma, cuando estemos trabajando con un procesador nativo de 8 bits, lo mejor será elegir tipos de datos de 8 bits.

Ahora que revisamos los tipos de datos que tenemos a nuestra disposición, es tiempo de conocer los operadores que nos permitirán trabajar con ellos. En la siguiente tabla se muestran, a grandes rasgos, los tipos de operadores disponibles. Más adelante analizaremos cada uno en detalle:

Tipo de operador	Ejemplos de operadores	Descripción
Aritméticos	**Asignación**	Se trata de la forma de entregar o asignar un valor.
	+, -, *, /	Operaciones aritméticas básicas: suma, resta, multiplicación y división.
	Módulo	Obtiene el resto de la división de un número por otro.
Compuestos	**++**	Los operadores compuestos permiten abreviar el código, por ejemplo, incrementar en uno el valor anterior.
	+= , - =, *=, /=	Estos operadores compuestos equivalen a x = x + y, x = x - y, x = x * y, y x = x / y, respectivamente.
Comparación	**==, !=, <, >, <=, >=**	Este tipo de operadores nos permiten comparar dos datos, por ejemplo, menor que, mayor que, diferente de, etcétera.
Booleanos	**AND (&&), OR (||) y NOT (!)**	También se conocen como operadores lógicos. Nos permiten conectar dos datos mediante el uso de nexos lógicos, comparando dos expresiones para devolver VERDADERO o FALSO.

Operadores aritméticos

Los operadores aritméticos son aquellos que más fácilmente asociamos con operaciones matemáticas; en este grupo se encuentran las cuatro operaciones fundamentales y también la asignación y el módulo.

- **+** operador suma (x=y+20)
- **-** operador resta (x=y-10)
- ***** operador multiplicación (x=y*2)
- **/** operador división (x=y/3)
- **=** operador de asignación (int numero1=25)

El operador **%** o módulo se encarga de devolver el resto cuando un número entero es dividido por otro. Por ejemplo, en **x = 7 % 5** obtendremos como resultado **2**.

Las operaciones, como la suma o la división, tendrán en cuenta el tipo de dato que hemos asignado. Por ejemplo, si ejecutamos la operación 9/4, pero hemos indicado que los datos son **int**, obtendremos como resultado 2, no 2.25 pues el tipo de dato **int** no reconoce decimales. Si necesitamos obtener resultados con decimales, será necesario elegir el tipo de dato **float**.

Operadores compuestos

Una operación compuesta integra una operación aritmética junto a una variable asignada. Entre estos operadores, encontramos el incremento (++) y el decremento (--), que se utilizan para aumentar o disminuir el valor almacenado en una variable, respectivamente.

Podemos usarlos de la siguiente forma:

- **X++** para incrementar x una unidad y devolver el antiguo valor de x.
- **x--** para decrementar x una unidad y devolver el antiguo valor de x.
- **++X** para incrementar x una unidad y devolver el nuevo valor de x.
- **--X** para decrementar x una unidad y devolver el nuevo valor de x.

Otros operadores compuestos no son más que una simplificación de operaciones complejas y sirven para realizar una operación matemática en una variable con otra variable o constante. Por ejemplo, si escribimos **x+=y**, equivale a la expresión **x = x + y** , o si escribimos **x- =y**, es igual a la expresión **x = x – y**.

Operadores de comparación

Utilizaremos los operadores de comparación en forma frecuente cuando trabajemos con estructuras condicionales, pues nos permiten averiguar si una condición es verdadera. A continuación, presentamos los operadores de comparación:

- **>** mayor que
- **<** menor que
- **>=** mayor o igual que
- **<=** menor o igual que
- **==** igual que
- **!=** diferente

El resultado de estos operadores puede ser **TRUE** (verdadero) o **FALSE** (falso), su importancia radica en que, aplicados en estructuras condicionales, nos permitirán determinar el camino que debe seguir nuestro sketch, dependiendo de si la operación indicada resulta verdadera o falsa. Por ejemplo, podríamos verificar el nivel de tensión en un pin de la tarjeta Arduino y, según la lectura, activar o apagar un led.

Algunos ejemplos de operaciones de comparación son los siguientes:

```
60 < 120;

30 > 28;

45 <= 3 2;

20 == 21;
```

Las dos primeras comparaciones entregarán como resultado **TRUE**, mientras que las demás serán **FALSE.**

Igualdad y asignación

Aunque parezcan similares, los operadores == y = son diferentes. El primero, el signo igual escrito dos veces (==), corresponde a la operación de igualdad que nos permite comparar dos valores, devolviendo **TRUE** si son iguales y **FALSE** si son distintos. Por otra parte, el signo igual escrito una sola vez (=) se refiere a la operación que nos permite asignar un valor a una variable o constante.

Operadores lógicos

Cuando necesitamos realizar un análisis sobre más de alguna condición para que se produzca un resultado, es necesario utilizar los operadores lógicos. Los operadores lógicos básicos son los siguientes:

► **AND (&&):** permite validar dos o más condiciones, estas deben cumplirse todas para que se ejecute el código dentro de las llaves. Por ejemplo:

```
if (variable1> variable2 && variable3 > variable4) {

digitalWrite(pinLedRojo,HIGH);

}
```

Si el valor de la variable1 es mayor que el almacenado en la variable2, y el valor de la variable3 es mayor que el de la variable4, se ejecutarán las instrucciones entre corchetes.

► **OR(II):** permite que, al cumplirse cualquiera de las condiciones, se ejecute el código entre las llaves. Por ejemplo:

```
if (variablel > variable2 || variable3 > variable4) {

digitalWrite(pinLedVerde, HIGH);

}
```

Si el valor de la variable1 es mayor que el almacenado en la variable2, o el valor de la variable3 es mayor que el de la variable4, se ejecutarán las instrucciones entre corchetes.

► **NOT (!):** este operador ejecuta las sentencias que están entre llaves cuando la condición es evaluada como falsa.

Práctica 2. Secuencia con siete leds

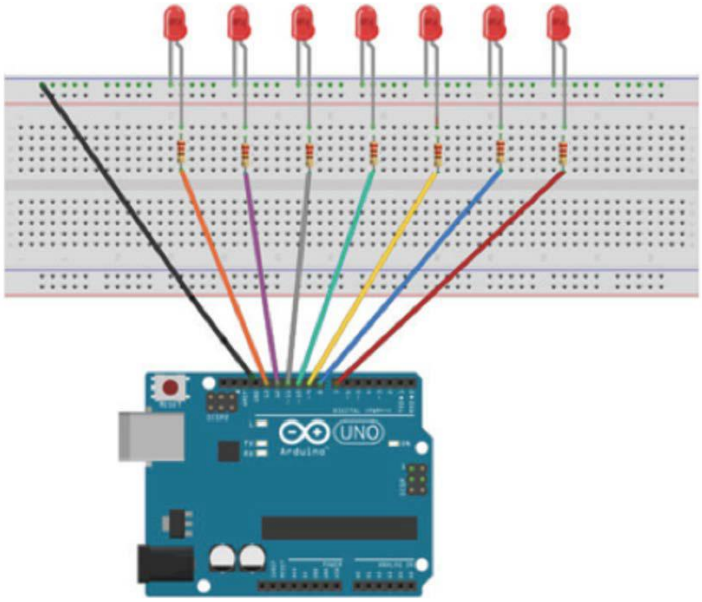
Instrucciones: Lee con atención lo que se indica en la siguiente práctica, cumpliendo con cada uno de los puntos que se especifican.

Objetivos:

- Se realizará un circuito en el que los leds se enciendan y se apaguen simulando el efecto de una estela de luz. Es decir, se programará una secuencia de encendido y apagado para cada led, uno después del otro, para recrear tal efecto.

- Una vez que la estela llegue al final, deberá volver, haciendo el recorrido inverso.

Esquema de conexión:



Lista de materiales:

Placa Arduino.
Protoboard.
Cable USB.
Siete resistencias de 220 Ohms.
Siete leds.

Cable conexión.

Código de la práctica.

```
/*Código para la práctica 2. Secuencia de 7 leds*/
int pin_rojo_1=13;  //asignamos el pin digital 13 al led rojo 1
int pin_rojo_2=12;  //asignamos el pin digital 12 al led rojo 2
int pin_rojo_3=11;  //asignamos el pin digital 11 al led rojo 3
int pin_rojo_4=10;  //asignamos el pin digital 10 al led rojo 4
int pin_rojo_5=9 ; //asignamos el pin digital 9 al led rojo 5
int pin_rojo_6=8; //asignamos el pin digital 8 al led rojo 6
int pin_rojo_7=7; // asignamos el pin digital 7 al led rojo 7
void setup() {
        pinMode(pin_rojo_1,OUTPUT); //declaramos que el pin 13 será de salida
        pinMode(pin_rojo_2,OUTPUT); //declaramos que el pin 12 será de salida
        pinMode(pin_rojo_3,OUTPUT); //declaramos que el pin 11 será de salida
        pinMode(pin_rojo_4,OUTPUT); //declaramos que el pin 10 será de salida
        pinMode(pin_rojo_5,OUTPUT); //declaramos que el pin 9 será de salida
        pinMode(pin_rojo_6,OUTPUT); //declaramos que el pin 8 será de salida
        pinMode (pin_rojo_7,OUTPUT); //declaramos que el pin 7 será de salida
        digitalWrite(pin_rojo_1,LOW); //nos aseguramos de que el led rojo 1 empezará
                                      //apagado
        digitalWrite(pin_rojo_2,LOW); //nos aseguramos de que el led rojo 2 empezará
                                      //apagado
        digitalwrite(pin_rojo_3,LOW); //nos aseguramos de que el led rojo 3 empezará
                                      //apagado
        digitalwrite(pin_rojo_4,LOW); //nos aseguramos de que el led rojo 4 empezará
                                      //apagado
        digitalWrite(pin_rojo_5,LOW); //nos aseguramos de que el led rojo 5 empezará
                                      //apagado
        digitalWrite(pin_rojo_6,LOW); //nos aseguramos de que el led rojo 6 empezará
                                      //apagado
        digitalWrite(pin_rojo_7,LOW); //nos aseguramos de que el led rojo 7 empezará
                                      //apagado
}
void loop() {
        //********Iniciamos la secuencia de ida******
        digitalWrite(pin_rojo 1,HIGH); //empezamos la secuencia de intermitencia
                                       //activando el led 1
        delay(1000); // esperamos 1 segundo con el led 1 activado
        digitalwrite(pin_rojo_1,LOW); //desactivamos el led rojo 1
        digitalwrite(pin_rojo_2,HIGH); //activamos el led rojo 2
```

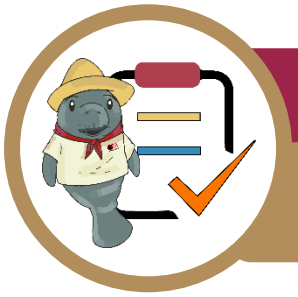
```
delay(1000); // esperamos 1 segundo con el led 2 activado
digitalWrite(pin_rojo_2,LOW); //desactivamos el led rojo2
delay(1000); // esperamos 1 segundo con el led rojo 2 desactivado
digitalWrite(pin_rojo_3,HIGH); //activamos el led rojo 3
delay(1000) ; //esperamos 1 segundo con el led 3 activado
digitalWrite(pin_rojo_3,LOW); //desactivamos el led rojo 3
digitalwrite(pin_rojo_4,HIGH); //activamos el led rojo 4
delay(1000) ; //esperamos 1 segundo con el led 4 activado
digitalWrite(pin_rojo_4,LOW); //desactivamos el led rojo 4
delay(1000) ; //esperamos 1 segundo con el led rojo 4 desactivado
digitalwrite (pin_rojo_5,HIGH); //activamos el led rojo 5
delay(1000) ; //esperamos l segundo con el led 5 activado
digitalWrite(pin_rojo_5,LOW); //desactivamos el led rojo 5
digitalWrite pin_rojo_6,HIGH); //activamos el led rojo 6
delay(1000) ; //esperamos 1 segundo con el led 6 activado
digitalWrite(pin_rojo_6,LOW); //desactivamos el led rojo 6
delay(1000); //esperamos l segundo con el led rojo 6 desactivado
digitalWrite(pin_rojo_7,HIGH); //activamos el led rojo 7
delay(1000); // esperamos 1 segundo con el led 7 activado
//******Iniciamos la secuencia de vuelta ********
digitalWrite(pin_rojo_7,LOW); //desactivamos el led rojo 7
digitalWrite(pin_rojo_6,HIGH); //activamos el led rojo 6
delay(1000); //esperamos 1 segundo con el led 6 activado
digitalWrite pin_rojo_6,LOW); //desactivamos el led rojo 6
delay(1000); //esperamos 1 segundo con el led rojo 6 desactivado
digitalwrite(pin_rojo_5,HIGH); //activamos el led rojo 5
delay (1000); //esperamos 1 segundo con el led 5 activado
digitalWrite(pin_rojo_5,LOW); //desactivamos el led rojo 5
digitalWrite(pin_rojo_4,HIGH); //activamos el led rojo 4
delay(1000); // esperamos 1 segundo con el led 4 activado
digitalWrite(pin_rojo_4,LOW); //desactivamos el led rojo 4
delay(1000); // esperamos 1 segundo con el led rojo 4 desactivado
digitalwrite(pin_rojo_3,HIGH); //activamos el led rojo 3
delay(1000); //esperamos 1 segundo con el led 3 activado
digitalWrite(pin_rojo_3,LOW); //desactivamos el led rojo 3
digitalwrite(pin_rojo_2,HIGH); //activamos el led rojo 2
delay(1000); // esperamos 1 segundo con el led 2 activado
digitalWrite(pin_rojo_2,LOW); //desactivamos el led rojo 2
delay(1000); //esperamos 1 segundo con el led rojo 2 desactivado
digitalWrite(pin_rojo_l,HIGH); //empezamos la secuencia de intermitencia
                                //activando el led 1
```

```
delay(1000); // esperamos 1 segundo con el led 1 activado
digitalWrite(pin_rojo_l,LOW); //desactivamos el led rojo 1
// el proceso vuelve a comenzar
}
```

Copia este código al IDE de Arduino y súbelo a la placa, conecta todos los componentes de acuerdo al diagrama proporcionado anteriormente y contesta las siguientes preguntas:

- ¿Qué tipos de datos se utilizaron en el código?
- ¿Cuáles fueron las instrucciones utilizadas en el código de la secuencia de los 7 leds?
- ¿Pudiste deducir par que sirve cada una de ellas? (especifica)
- ¿Conoces alguna instrucción para hacer más corto el código? ¿Cuál y como la utilizarías?

Realiza un reporte de la práctica debidamente estructurado y entrega al docente.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 3

Práctica 2: Secuencia con siete leds

DATOS GENERALES						
Nombre(s) del alumno (s)				**Matriculas(s)**		
Producto: Reporte de práctica				**Fecha**		
Formación laboral básica: Robótica **Submódulo 1:** Microcontroladores y microprocesadores				**Periodo: 2026 – 2027A**		
Nombre del docente:				**Firma del docente**		

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Copió correctamente el código al IDE de Arduino.			2		
2.	El código se compila sin errores y es funcional.			1		
3.	El circuito físico está bien conectado y corresponde al esquema.			1		
4.	La secuencia de LEDs funciona correctamente en ambos sentidos (ida y vuelta).			2		
5	Responde claramente a las preguntas de análisis (tipos de datos, instrucciones, deducción, mejora).			1		
6	Presenta un reporte bien estructurado con redacción clara.			2		
				CALIFICACIÓN		

Lectura 7. Estructuras de control

Este tipo de instrucciones permiten al programador controlar las acciones y decisiones que debe tomar Arduino frente a un problema, ya sea un robot autónomo móvil o ya sea para realizar un proyecto de demótica en el que se desee subir o bajar una persiana automáticamente.

Estas instrucciones nos permiten controlar el flujo de datos que van siendo capturados o generados por Arduino y sus sensores, actuadores o componentes electrónicos.

Bifurcaciones

En programación podemos definir bifurcación como la decisión de tomar un camino u otro dentro de la ejecución de un programa.

La decisión de seguir un camino o seguir otro camino viene dada por la evaluación de una condición, que, dependiendo del tipo de instrucción de bifurcación que sea, será una comprobación de una comparación con TRUE o FALSE o por el valor propiamente dicho del elemento utilizado en dicha comprobación.

Independientemente del tipo de bifurcación que estés utilizando, tienes que especificar la condición a evaluar, y además, agrupar el conjunto de instrucciones que forman cada camino posible a tomar en el camino correspondiente indicando el inicio y el fin de cada camino.

Cada lenguaje de programación define las bifurcaciones de forma distinta, pero la utilización y los componentes son los mismos para todos los lenguajes.

if / else / elseif

La sentencia **if** (si condicional**)** nos permite controlar o redirigir la ruta que debe seguir el programa. Es decir, evaluará un dato contenido en una variable y decidirá si la condición se cumple o no.

Por ejemplo: Si la variable A almacena el número 3, ejecuta las siguientes instrucciones ...

Si la variable A no contiene el número 3, este bloque de programa lo obviará y seguirá justo donde acaban las instrucciones que no se van a ejecutar.

Veamos un ejemplo:

```
if (A==3 ) { // si A es igual a 3, haz lo siguiente. ..
A=A+5; // A almacenará el resultado de 3+5
C=5+4; // C almacenará el resultado de 5+4
```

```
B=A+C; // B almacenará el resultado de sumar A+C
}
C=4;  //si la condición no se cumple (A=3), C almacena un 4
```

Es fácil seguir el ejemplo, si leemos los comentarios que tenemos en cada línea del código. Cuando la condición no se cumple, Arduino salta todo el bloque del **if** y pasa directamente a ejecutar la línea: C=4; //si la condición no se cumple (A=3), C almacena un 4.

De esta forma, podemos dotar a nuestros proyectos sobre robótica o domótica del «privilegio» de escoger entre una condición u otra en función de un valor determinado, en el caso del ejemplo A=3.

La instrucción **else** (si no…) suele ir acompañada de un **if**. Juntas se pueden interpretar como: **si** la condición se cumple, realiza lo que se indica; **si no**, realiza esto otro.

Veamos un ejemplo:

```
i f (A==3) { // si A es igual a 3, haz lo siguiente ...
A=A+5; // A almacenará el resultado de 3+5
C=5+4; // C almacenará el resultado de 5+4
B=A+C; // B almacenará el resultado de sumar A+C
}
else {
C=4 ; //si no, C almacena un 4
}
```

Observemos que en el bloque **else** podríamos añadir otras sentencias, y definir así un bloque de nuevas sentencias que ejecutar si la condición del **if** no se cumpliese.

Cuando trabajamos con este tipo de estructuras de control (**if**-**else**), es posible detallar aún más las instrucciones que se ejecutarán, para ello integraremos **elseif**. La función de esta modificación es permitirnos detallar el control que ocurre en el bloque **if**. Gracias a **elseif**, introduciremos líneas de Código adicionales dentro de if, teniendo la oportunidad de ejecutar un bloque de acciones alternativo, por ejemplo:

```
if (variable1 < variable2){ // Si variable1 es menor que variable2
  digitalWrite(PinLedRojo,HIGH); // enciende PinLedRojo
}
elseif (variable1==variable2) { //pero si variable1 es igual a variable2
  digitalWrite(PinLedVerde, HIGH); // enciende PinLedVerde
}
else { //sino ocurre nada de lo anterior
  digitalWrite(PinLedRojo,LOW); //apaga PinLedRojo
}
```

En este ejemplo, encenderemos el led rojo en caso de que la variable1 sea menor a la variable2, pero encenderemos el led verde si ambas variables son iguales. Si ninguno de estos tests resulta verdadero, apagaremos el led rojo.

Switch / case / break

La sentencia **switch** te va a permitir crear diferentes caminos de ejecución posibles en función de un valor. Básicamente, la bifurcación **switch** funciona de forma parecida a como lo hace la bifurcación con **if/else** anidados, pero, añadiendo cierta funcionalidad adicional.

Cuando Arduino se encuentra con un bloque formado por estas instrucciones, actúa de la siguiente manera:

1. Se compara el valor de la variable que posee switch (entre paréntesis) con el valor de una variable del programa que figura en una instrucción case.
2. Si el valor de la variable es igual a algunos de los valores que contiene la instrucción case, se ejecuta esa parte del código.

Veamos un ejemplo para una mejor comprensión del concepto:

```
char vocal = LeerCaracter();
switch(vocal)
{
 case 'a':
  SentenciasA();
  break;
 case 'e':
  SentenciasE();
  break;
 case 'i':
  SentenciasI();
  break;
 case 'o':
  SentenciasO();
  break;
 case 'u':
  SentenciasU();
  break;
 default:
  SentenciasNoVocal();
}
```

La sentencia "break" y "default"

La sentencia **break** se emplea para devolver la secuencia del programa a la siguiente instrucción que se va a ejecutar después de haber entrado en la estructura **switch**. Si prescindiéramos de **break**, el programa se quedaría en ese punto, esperando la sentencia **break** para seguir con el programa.

Observemos que, si no se cumple con ninguna condición, por defecto el programa pasa a **default**, ejecutándose las sentencias que contiene.

En el ejemplo se ha utilizado una bifurcación switch para evaluar la lectura de un carácter. Los posibles caminos que se pueden tomar son los valores de las vocales, cada uno ejecutará su conjunto de sentencias correspondiente, y para aquellos valores diferentes de las 5 vocales se ejecutará el conjunto de instrucciones correspondiente a los caracteres no vocales.

Bucles

En programación podemos definir un bucle como la repetición de la ejecución de un conjunto de instrucciones. Cada repetición del conjunto de instrucciones se llama iteración.

En programación existen diferentes tipos de bucles, y debes tener muy claro cuándo usar un bucle o cuándo usar otro, ya que cada uno de ellos está recomendado para ser usado en diferentes contextos, que varían dependiendo del tipo de condición de parada de ejecución del bucle que se necesite, en otras palabras, utilizarás un bucle u otro dependiendo de la forma en la que necesites indicarle el número de iteraciones a realizar.

Independientemente de que tipo de bucle estés utilizando se debe indicar el punto de inicio, el punto final y el número de iteraciones que van a realizarse, aunque para cada tipo de bucle se especifica de una forma distinta, todos tienen estos elementos comunes.

Cada lenguaje de programación define los bucles de forma distinta, pero la utilización y los componentes son los mismos para todos los lenguajes.

for

Esta instrucción nos permite repetir un número determinado de veces una o un conjunto de sentencias.

Puede servirnos de contador para una variable o como bucle de retardo para ejecutar una serie de instrucciones.

Veamos un ejemplo:

```
for (int f=O; f<lO; f++) {
A=A+5; // A inicialmente almacena un cero
}
```

Observemos que dentro de la sentencia **for**, entre paréntesis y delimitado por punto y coma, tenemos tres partes diferenciadas que hacen posible el bucle.

La primera parte declara ahí mismo la variable f como entero, y le asignamos el valor de inicio cero para llevar la cuenta del número de iteraciones que se hacen.

La segunda parte introduce la condición para que el bucle se repita. En este caso, el número de iteraciones que hay que dar es 10, y viene dado por "f<10", ya que esta variable (f) se inicializó en 0.

La tercera parte incrementa en una unidad a la variable f hasta que llegue a 10, donde se cumplirá la condición expuesta en la segunda parte, y el bucle finalizará.

Si analizamos el bucle creado en el ejemplo, la variable A acabará almacenando el número 50, ya que va sumándole cada vez 5.

Incremento del "for"

El incremento de la instrucción **for**, se puede dar en una cantidad diferente de 1 (**f++**), por ejemplo: Para incrementar un bucle **for** de dos en dos en Arduino, se debe modificar la parte de "incremento" del bucle. En lugar de **f++**, se debe usar **f += 2** o **f = f + 2**.

El operador **+=** es una forma abreviada de sumar un valor a una variable y asignar el resultado a la misma variable. Es lo mismo que **f = f + 2**. En este caso, se suma 2 a la variable **f** y se guarda el resultado de vuelta en **f**.

While (mientras…)

Los bucles **while** se utilizan cuando no se sabe exactamente el número de iteraciones que tiene el bucle, pero sí que se sabe que mientras que se cumpla una condición el bucle debe de seguir ejecutándose, es decir, en los bucles **while** se busca ejecutar un conjunto de instrucciones de forma repetitiva mientras se cumpla una condición.

En los bucles **while** se utiliza una comparación que en caso de ser TRUE continuará con la ejecución, mientras que si el resultado es F ALSE parará la ejecución del bucle. La comparación es igual que la que te hemos explicado con la sentencia if, y es comprobada en cada iteración del bucle.

Al contrario que en los bucles **for**, en los bucles **while** no existe una variable que se utilice para saber el número de iteraciones que se llevan ejecutadas, ya que no es necesario saberlo como condición de parada del bucle.

En el comienzo del bucle se encuentra la comparación que sirve como punto de entrada al bucle, por tanto, si la comparación al empezar es FALSE el bucle jamás se ejecutará.

En los bucles **while** debes tener en cuenta lo siguiente:

- Antes de comenzar la ejecución debes de inicializar las variables que vayas a utilizar dentro de la comprobación.

- Al acabar cada iteración debes de modificar los valores necesarios que estén presentes en la comprobación y que hayan sido alterados durante la ejecución de la iteración. En otras palabras, cada iteración debe modificar los valores de las variables utilizadas en la comprobación buscando que en algún momento la comprobación pase a tener valor FALSE.

Si no tienes en cuenta estos dos puntos podrías encontrarte con los siguientes bucles:

- En el caso de no tener en cuenta la correcta inicialización de los componentes de la comprobación podrías crear bucles que nunca se ejecuten, es decir, bucles que desde el principio la comprobación es FALSE.

- En el caso de no tener en cuenta la alteración de los valores de los componentes de la comprobación podrías crear bucles infinitos, bucles en los que jamás la comprobación es FALSE.

A continuación, te mostramos un ejemplo de bucle **while**:

```
int posicion = 0;
bool encontrado = false;
bool cadenaacabada = false;
String cadena = "Hola mundo" ;
while(!encontrado && !cadenaacabada)
{
 if(posicion == cadena.length()) //length() es una función útil para saber
  cadenaacabada = true;        //cuántos caracteres tiene una cadena de texto
 else
 {
  if(cadena[posicion]=='a')
   encontrado = true;
  else
   posicion = posicion+1;
 }
}
```

La palabra reservada para crear este tipo de bucles es **while**, y acto seguido, entre paréntesis se introduce la condición de ejecución del bucle, y posteriormente el conjunto de sentencias que lo componen. Tal y como puedes ver, antes de iniciar el bucle se han inicializado las variables que se utilizarán durante la ejecución de este.

En el ejemplo se ha programado un código fuente que va a buscar la primera ocurrencia de un carácter concreto en una cadena de texto. La condición de ejecución del bucle viene determinada por dos variables de tipo bool, la primera de ellas indica que el bucle parará si ha encontrado el carácter indicado y la segunda de ellas indica que el bucle parará si se ha llegado al final de la cadena de texto. En la ejecución del bucle puedes comprobar que dichas variables se pondrán a TRUE cuando se llegue al final o cuando se encuentre el carácter, en caso contrario aumentará el valor de lectura del String para pasar a leer el siguiente carácter en la siguiente iteración.

do-while (hacer mientras…)

Los bucles **do** se utilizan cuando se quiere realizar una primera iteración y después comprobar si seguir ejecutando o no.

La gran diferencia entre los bucles **for** y **while** con los bucles **do** es que la condición de comprobación en estos últimos se encuentra al final de este. Por tanto, no se saben a priori el número de iteraciones que se ejecutarán en el bucle, lo único que se puede asegurar es que al menos se realizará una iteración.

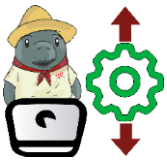
La condición de salida del bucle indicará si se tiene que continuar ejecutando el bucle o no, en este sentido funciona igual que los bucles **while**, pero al final de la iteración. Al acabar una iteración, si la comprobación que se realiza es FALSE el bucle parará, en cambio, si la comprobación es TRUE el bucle ejecutará una nueva iteración.

Ejemplo del uso del bucle **do**:

```
int valorleido = 0;
do
{
valorleido = leerSensor();
delay(300);
}
while (valorleido < 50);
```

Las palabras reservadas para crear estos bucles son **do** para indicar el comienzo del bucle y **while** para indicar la condición de salida del bucle.

En el ejemplo se está utilizando una función para realizar la lectura de un sensor, y en función del valor leído acabará el bucle, concretamente cuando el valor leído sea 50 o más.



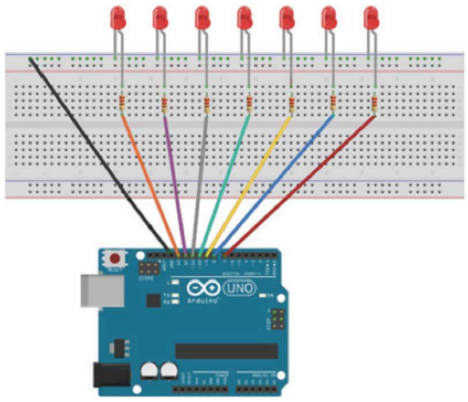
Práctica 3. Secuencia con siete leds simplificado

Instrucciones: Lee con atención lo que se indica en la siguiente práctica, cumpliendo con cada uno de los puntos que se especifican.

Objetivos:

- Se realizará un circuito en el que los leds se enciendan y se apaguen simulando el efecto de una estela de luz. Es decir, se programará una secuencia de encendido y apagado para cada led, uno después del otro, para recrear tal efecto.
- Una vez que la estela llegue al final, deberá volver, haciendo el recorrido inverso.
- Reducir el código de la práctica 2 al utilizar las estructuras de control adecuadas.

Esquema de conexión:



Lista de materiales:

"Los materiales son los mismos que los que se utilizaron en la práctica 2"

Código de la práctica.

Como pudiste observar, el código de la práctica 2 es demasiado largo, y seguramente te planteaste la siguiente pregunta: ¿Y si en vez de siete leds son doscientos los que deseamos incorporar a nuestro circuito? La respuesta es obvia: sería una tarea muy dura de realizar.

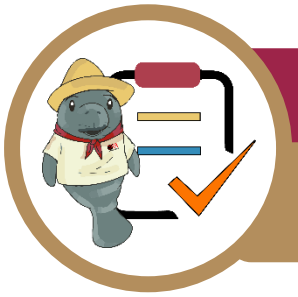
A continuación, se propone una nueva versión del código para esta práctica.

```
/*Código para la práctica 3. Secuencia de 7 leds simplificado*/
Void setup() {
        for (int i=7; i<=13; i++) {
        pinMode (i,OUTPUT); //indicamos que se asignen como salida los pines desde el 7
                                //hasta el 13
        }
}
void loop() {
        for(int i=7; i<=13; i++) {
        //indicamos que repita el siguiente proceso desde 7 hasta 13, que son los pines
       //escogidos para los leds
        digitalWrite(i,HIGH); //activamos los leds por orden según el bucle
        delay(1000);  //Cada led estará activo durante 1 segundo
        digitalWrite(i,LOW); //desactivamos los leds por orden según el bucle
        }
        for(int i=13; i>=7; i--) {
        //indicamos que repita el siguiente proceso desde 13 hasta 7, que son los pines escogidos
        //para los leds, pero ahora en sentido contrario.
        digitalWrite(i,HIGH); //activamos los leds por orden según el bucle
        delay(1000); //Cada led estará activo durante 1 segundo
        digitalWrite(i,LOW); //desactivamos los leds por orden según el bucle
        }
}
```

Copia este código al IDE de Arduino y súbelo a la placa, conecta todos los componentes de acuerdo al diagrama proporcionado anteriormente y contesta las siguientes preguntas:

- ¿Qué tipos de datos se utilizaron en el código?
- ¿Cuáles fueron las instrucciones utilizadas en el código de la secuencia de los 7 leds simplificado?
- ¿Cuál fue la instrucción auxiliar para que el código de la práctica 2 se simplificara?

Realiza un reporte de la práctica debidamente estructurado y entrega al docente.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 4

Práctica 3: Secuencia con siete leds simplificado

DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Reporte de práctica		**Fecha**	
Formación laboral básica: Robótica **Submódulo 1:** Microcontroladores y microprocesadores		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	El estudiante conecta correctamente el circuito según el esquema indicado.			2		
2.	El código fue copiado y cargado correctamente al IDE de Arduino.			1		
3.	El funcionamiento del circuito refleja correctamente la estela de luz ida y vuelta.			2		
4.	Identifica correctamente los tipos de datos utilizados en el código.			1		
5	Menciona correctamente las instrucciones utilizadas para la secuencia de LEDs.			1		
6	Señala correctamente la instrucción que permite la simplificación del código.			1		
7	Entrega un reporte claro, estructurado y completo de la práctica.			2		
				CALIFICACIÓN		

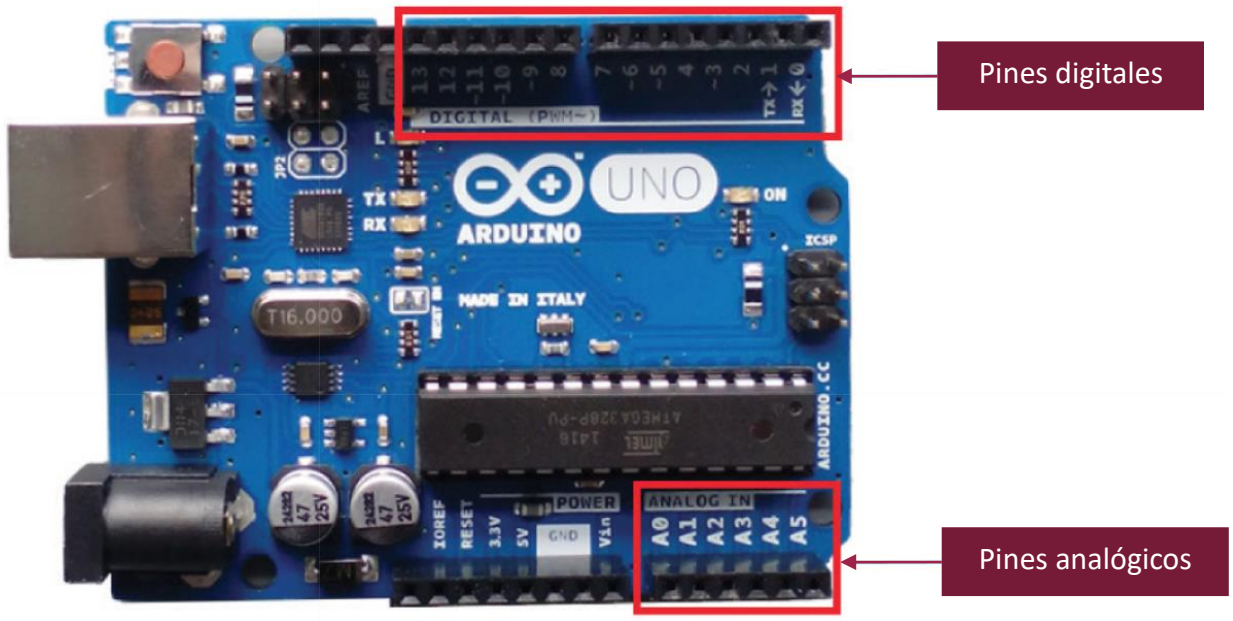
Lectura 8. Configuración de puertos de entrada y salida

Arduino es una plataforma que nos permite interactuar con el medio ambiente gracias a la multitud de sensores, módulos y complementos que posee.

Para poder decirle a Arduino que deseamos adquirir cierta información mediante un sensor, debemos configurar los pines de que dispone para tal efecto.

Antes de determinar si el pin deberá adquirir información analógica o digital, deberemos declarar si deseamos que sea de entrada o de salida, es decir, si nos interesa que introduzca datos en Arduino o los extraiga.

Las funciones se deberán declarar en el bloque **void setup()**. Por tanto, definiremos un pin de salida como aquel mediante el cual deseamos que los datos procesados por Arduino salgan hacia otro dispositivo (**OUTPUT**). Y definiremos un pin de entrada como aquel mediante el cual deseamos que los datos sean introducidos por él desde un dispositivo externo hacia Arduino para su procesamiento posterior (**INPUT**).



Para llevar a cabo este ´proceso, se utiliza la instrucción **pinMode**.

pinMode()

Vemos su sintaxis:

```
pinMode (pin, INPUT/OUTPUT);
```

pin: será el pin que hace referencia a la placa Arduino donde conectaremos el sensor u otro dispositivo.

- INPUT: definiremos el pin anterior como pin de entrada. Adquisición de datos por parte de Arduino para su posterior tratamiento.

- OUTPUT: definiremos el pin anterior como pin de salida. Extracción de datos por parte de Arduino hacia un dispositivo externo.

Por ejemplo:

```
/*Código ejemplo para definir el pin 13 como pin de salida*/
int pin_de_salida = 13;
void setup() {
  pinMode (pin_de_salida,OUTPUT);
}
void loop() {
.
.
.
}
```

Hay que destacar que declaramos la variable pin _de_salida y la vinculamos al pin 13 de Arduino; más adelante, declararemos que este pin será de salida.

Podríamos haber procedido de la siguiente manera:

```
/*Código ejemplo para definir el pin 13 como pin de salida*/
void setup () {
pinMode (13, OUTPUT) ;
}
void loop () {
.
.
.
}
```

Como podemos ver, no se ha vinculado ninguna variable al pin 13, declarándolo directamente en la función **pinMode**.

Entrada y salida digitales

Arduino está dotado de una serie de pines de entrada y salida digitales. Esto quiere decir que la información que sea introducida o extraída será de carácter digital.

Cuando hablamos de información de carácter digital nos estamos refiriendo a que los datos son 1 o 0, alto o bajo, 5 voltios o 0 voltios.

Por tanto, estos pines serán idóneos para conectar sensores o dispositivos que trabajen en forma digital. Dicho de otro modo: si el sensor da información, enviará 5 voltios; si por el contrario el sensor no detecta actividad, enviará O voltios.

No todos los sensores trabajan o pueden trabajar de este modo, ya que hay algunos que proporcionan valores intermedios del 0 al 5. Estos sensores se conectarán a las entradas y salidas analógicas.

digitalWrite()

Cuando el pin sea digital y esté configurado como **OUTPUT** (salida), se empleará la función **digitalWrite** preestablecida en Arduino.

Veamos su sintaxis:

```
digitalWrite (pin, valor);
```

pin: pin digital configurado o establecido como **OUTPUT.**

valor: valor que deseamos transferir al pin, que será **HIGH** o **LOW.**

Un ejemplo:

```
/*Código ejemplo escribir en un pin digital*/
int pin_digital = 13; //pin digital
void setup() {
  pinMode (pin_digital, OUTPUT) ; //pin configurado como salida
}
void loop ( ) {
  digitalWrite (pin_digital, HIGH) ; //el pin nos proporciona 5V
}
```

Si en el ejemplo anterior se colocara un componente electrónico de salida, como un diodo led, éste se encendería, y emitiría luz después de ser leído el programa por el microcontrolador de Arduino.

Si en el programa anterior le cambiamos la constante HlGH por LOW, el diodo led deja de emitir luz.

Observemos el siguiente ejemplo:

```
/*Código ejemplo escribir en un pin digital*/
int pin_digital = 13; //pin digital
void setup() {
 pinMode (pin_digital,OUTPUT); //pin configurado como salida
}
void loop() {
 digitalWrite (pin_digital,HIGH); //nuestro pin nos proporciona 5V
 digitalWrite (pin_digital,LOW); //nuestro pin nos proporciona 0V
}
```

Como ejemplo, se ha escrito un programa para Arduino que activa y desactiva un diodo led, por lo que se ha logrado recrear un efecto de intermitencia del diodo led.

El único problema es que Arduino ejecuta las instrucciones muy rápidamente, y es muy probable que no tengamos tiempo de visualizar el encendido y apagado del diodo led.

Para solventar esta cuestión de tiempos existe una función de Gestión del tiempo en Arduino predefinida para tal efecto, y que es la que más se utiliza para aplicar retardos en la ejecución de las instrucciones de nuestros programas si así se requiere.

digitalRead()

La instrucción digitalRead realiza la lectura de un pin digital que está configurado como entrada de información a la placa.

Veamos su sintaxis:

```
digitalRead (pin);
```

pin: pin digital configurado o establecido como INPUT

El valor que lee esta función de un sensor normalmente se almacena en una variable que define el usuario. Es por esto que normalmente veremos la función **digitalRead()** precedida por una variable y un igual.

Si analizamos el siguiente ejemplo, su comprensión será más clara:

```
/*Código ejemplo leer desde un pin digital* /
int pin_digital = 13; //pin digital
int pin_led = 12; //un led irá conectado en el pin 12
int valor_sensor; //variable donde se guardarán los valores captados por el sensor
void setup() {
 pinMode (pin_digital,INPUT) ; //pin configurado como entrada
 pinMode (pin_led,OUTPUT) ; //pin configurado como salida
}
```

```
void loop() {
valor_sensor= digitalRead (pin_digital) ; //guardamos el valor proporcionado por
                                          el sensor en la variable "valor_ sensor"
if (valor sensor == HIGH) {   //si el valor es alto
digitalWrite (pin_led, HIGH) ;   //enciende el led
}
digitalWrite (pin_led, LOW) ;   //si no, no lo enciendas
}
```

En este ejemplo se han combinado las dos funciones digitales, es decir, la función de lectura y la función de escritura.

Como puede observarse, se declaran los dos pines en el bloque superior del programa, asignando un pin de entrada (pin 13) para el sensor y un pin de salida (pin 12) para un diodo led.

Mediante una sentencia condicional (SI condicional), cada vez que la variable *valor_sensor* esté activa o en estado alto (**HIGH**), el diodo led conectado al pin 12 emitirá luz, mientras que, de lo contrario, el led no se iluminará.

Esto se repetirá una y otra vez, siempre y cuando el valor del sensor sea alto.

Entrada y salida analógica

Arduino también está dotado de una serie de pines de entrada y salida analógicos. Podemos afirmar que la información que sea introducida o extraída será de carácter analógico.

Esto quiere decir que, a diferencia de la información digital (5 o 0 voltios), aquí nos podemos encontrar con que Arduino debe leer o escribir datos donde sus valores pueden oscilar o variar dentro del rango de 0 y 5 voltios.

Por tanto, estos pines serán idóneos para conectar sensores o dispositivos que al ser analógicos puedan dar cualquier valor comprendidos entre 0 y 5 voltios.

Como los valores comprendidos entre 0 y 5 voltios pueden ser muchos, Arduino pone el límite en 1024 valores. Dicho de otro modo, Arduino asignará un 0 para representar los 0 voltios, asignará el número 512 para representar 2.5 voltios y el número 1023 para los 5 voltios.

Un posible ejemplo de sensor que es susceptible de ser conectado en un pin analógico, es lo que se conoce como fotorresistencia o LDR. Una LDR varía su resistencia según la intensidad de la luz, estando los valores que proporciona comprendidos entre 0 y 1024 gracias a Arduino.

Otro ejemplo sería el sensor LM35. Se trata de un sensor de temperatura. Los valores que proporciona no serán 0 o 5 voltios (1 o 0, alto o bajo), sino que empleará un rango de valores según la temperatura, 1°, 8,5°, 25°, etc.

Como ocurre en el caso anterior de los pines digitales, las funciones que controlan a los pines analógicos son dos: **analogWrite()** y **analogRead()**.

analogWrite()

Cuando el pin sea analógico y esté configurado como OUTPUT (salida), emplearemos la función **analogWrite()**.

Veamos su sintaxis:

```
analogWrite (pin, valor) ;
```

Donde:

Pin: es el pin analógico configurado como OUT mediante la función **pinMode**.

Valor: valor entre 0 y 255. Siendo 0 el valor equivalente a 0 voltios y 255 el equivalente a 5 voltios.

Un ejemplo típico para comprender mejor esta función es el de variar la luminosidad de un led conectado a un pin analógico.

analogRead()

El valor que lee esta función de un sensor normalmente se almacena en una variable que define el usuario. Es por esto que normalmente veremos la función **analogRead()** precedida por una variable y un igual.

Sintaxis:

```
analogRead (pin);
```

pin: pin analógico configurado como INPUT.

Veamos un ejemplo:

```
/* Parte del código de un programa para medir la temperatura
con un sensor LM35*/
int temp= A0 ;   // asociamos el pin analógico A0 con la variable temp
void setup () {
  pinMode (temp, INPUT) ;  //configuramos el pin A0 (temp) como salida
}
void loop () {
  valor = analogRead(temp) ;   //la variable valor almacena el dato que sale por el
.                                        //pin leído A0 (temp)
.
.
.
}
```

Lectura 9. Gestión del tiempo en Arduino

Existen en el lenguaje de programación de Arduino algunas funciones relacionadas con el tiempo que nos permiten aplicar tiempos en la ejecución de instrucciones.

delay()

La función **delay** (tiempo en milisegundos) se utiliza para retardar la siguiente instrucción que se va a ejecutar en un programa Arduino.

Sintaxis:

```
delay (milisegundos) ;
```

Donde:

Milisegundos: el tiempo que va a esperarse hasta pasar a la siguiente instrucción de nuestro programa.

Por ejemplo:

delay (1000) ;

La función delay hará esperar 1 segundo al microcontrolador antes de pasar a la siguiente instrucción.

Hay que recordar que:

1 segundo = 1000 milisegundos = $1*10^{-3}$ segundos

Veamos ahora el ejemplo siguiente, añadiendo la función delay:

```
/*Código ejemplo de encendido y apagado de un diodo
led*/
int pin_digital = 13;  //pin digital
void setup() {
 pinMode (pin_digital, OUTPUT) ;   //pin configurado como salida
}
void loop() {
 digitalWrite (pin_digital, HIGH) ;  //nuestro pin nos proporciona 5V
 delay (1000) ;    //esperamos 1 segundo a que se apague el led
 digitalWrite (pin_digital, LOW) ;  //nuestro pin nos proporciona 0V
}
```

Estas instrucciones, al estar en el bloque «void loop», se repetirán continuamente.

El inconveniente que aparece con esta función es que el microcontrolador se detiene durante el tiempo -en milisegundos- que introducimos entre paréntesis. Esto hace que se pierda un tiempo excesivo, durante el cual el microcontrolador no está realizando nada.

millis()

La función **millis()** actúa de contador en el momento que es activada, por lo que contará en milisegundos desde el momento que es activada hasta aproximadamente 50 días.

Para que la función **millis()** devuelva el tiempo en milisegundos y los almacene en una variable, ésta debe ser del tipo *unsigned long*.

Veamos un ejemplo, para una mayor comprensión:

```
/* Ejemplo función millis */
unsigned long tiempo; //variable unsigned long
void setup() {
}
void loop() {
  tiempo = millis() ;  //la función se activa y se guarda en la variable tiempo
}
```

micros()

La función **micros()** también actúa de contador en el momento que es activada, por lo que contará en microsegundos desde el momento que es activada hasta unos 70 minutos.

Para que la función **micros()** devuelva el tiempo contabilizado y los almacene en una variable, ésta también debe ser del tipo *unsigned long*.

El ejemplo anterior para la función **millis()** también es aplicable para la función micros ().

delayMicroseconds()

Esta función responde de igual forma que la función **delay()**, sólo que entre paréntesis introduciremos el tiempo en microsegundos y no en milisegundos.

Un milisegundo equivale a 1000 microsegundos, y un segundo equivale a un millón de microsegundos. Independientemente de esto, el valor más alto que se puede utilizar en la función es 16 383 microsegundos. Toda parada que requiera ser mayor a ese valor deberá ser realizada utilizando la instrucción **delay()** en lugar de **delayMicroseconds()**.

Práctica 4. El botón del pánico

Instrucciones: Lee con atención lo que se indica en la siguiente práctica, cumpliendo con cada uno de los puntos que se especifican.

Objetivos:

- En esta práctica crearemos un circuito en el que, al pulsar un botón, un *buzzer* emitirá un tono intermitente; además, para reforzar la señal de alarma, un led se encenderá y apagará a la vez que el *buzzer* emita el sonido.

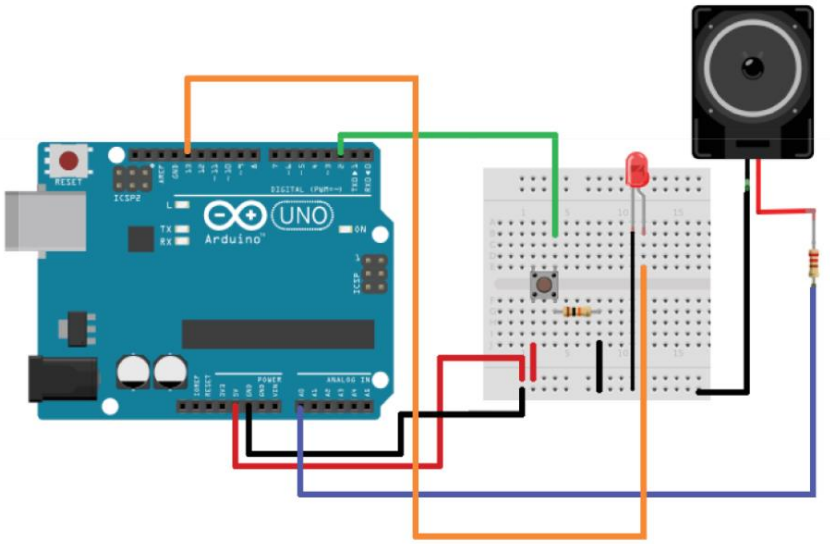
- Al pulsar dicho botón, la alarma sonará en diez ocasiones antes de detenerse.

Documentación bibliográfica:

Antes de la realización de la práctica, es necesario que conozcas o recuerdes el funcionamiento y características de algunos componentes que se utilizarán, es por ello que debes investigar lo siguiente:

- El botón
- El pulsador
- El interruptor
- Diferencias entre botón, pulsador e interruptor.
- El buzzer (función *tone())*

Esquema de conexión:



Lista de materiales:

- Placa Arduino.
- Protoboard.
- Cable USB.
- Una resistencia de 10 KOhms, o el valor más aproximado.
- Una resistencia de 220 Ohms para el zumbador.
- Un led.
- Un zumbador o buzzer.
- Un botón, o un pulsador.
- Cable conexión.

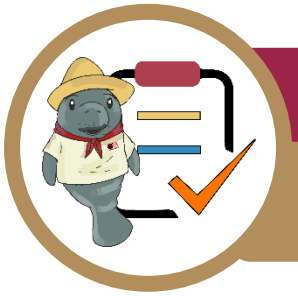
Código de la práctica.

```
/* Código Alarma mediante botón. Botón del pánico */
int buzzer=A0;
int boton=7;
int led=13;
int vb =0; //variable que almacena el valor del botón
void setup() {
        pinMode (buzzer, OUTPUT);
        pinMode (boton, INPUT) ;
        pinMode (led, OUTPUT);
}

void loop() {
        vb = digitalRead (boton);
        if (vb == HIGH) {
          for (int i=0; i<10; i++) { //el buzzer sonará 10 veces
          delay (220);
          tone(buzzer,220,100 );
          digitalWrite(led,HIGH);
          delay(220);
          noTone(buzzer);
          digitalWrite(led, LOW);
          }
        }
}
```

Copia este código al IDE de Arduino y súbelo a la placa, conecta todos los componentes de acuerdo con el diagrama proporcionado anteriormente y contesta las siguientes preguntas:

- ¿Qué tipos de datos se utilizaron en el código?
- ¿Cuáles fueron las instrucciones utilizadas en el código?
- ¿Qué tipo de entradas y salidas se utilizaron y cuáles fueron los componentes que se les asignaron?
- ¿Cuál fue la función de la instrucción *delay()* en relación con el contexto?

Realiza un reporte de la práctica debidamente estructurado y entrega al docente.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 5

Práctica 4: El botón del pánico

DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Reporte de práctica		**Fecha**	
Formación Laboral Básica: Robótica **Submódulo 1:** Microcontroladores y microprocesadores		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Conecta correctamente el circuito según el esquema indicado			2		
2.	Copia el código correctamente al IDE de Arduino y lo carga sin errores			1		
3.	El circuito ejecuta la función correctamente: suena el buzzer y se enciende el LED al pulsar botón			2		
4.	Describe correctamente los tipos de datos utilizados en el código			1		
5	Identifica correctamente las instrucciones utilizadas en el código			1		
6	Reconoce las entradas y salidas utilizadas y los componentes asociados			1		
7	Explica correctamente la función de la instrucción delay() en el contexto de la alarma			1		
8	Entrega un reporte claro, estructurado y completo de la práctica			1		
				CALIFICACIÓN		

Lectura 10. Modo de operación Standalone

Como ya sabes, una placa Arduino es una herramienta fantástica a la hora de realizar tus proyectos. Te permite crear el circuito que necesitabas y, lo que es aún mejor, personalizarlo.

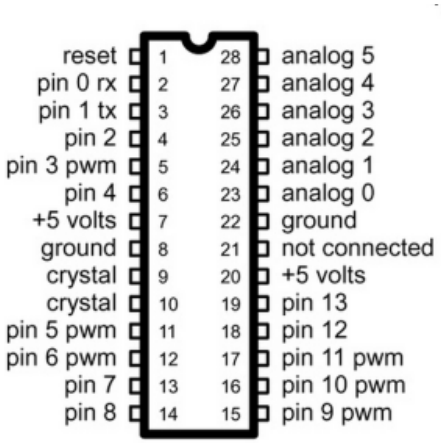
También puedes personalizar el propio Arduino, basta con conseguir los componentes y montarlo en función de tus necesidades. Puedes, desde cambiarle las luces LED por otras de distinto color, hasta eliminar el USB y hacer que se comunique con tu ordenador por un puerto serie. Una vez más, el límite aquí es tu imaginación.

Montar un Arduino UNO estándar sobre una protoboard recibe el nombre de **Standalone** y es especialmente útil si quieres dejar tu Arduino fijo en algún proyecto, pero no quieres perder la placa. Te permite reducir el espacio que ocupa, utilizar solo aquellos recursos que necesitas y ahorrar algo de dinero.

Vamos a ver cómo usar el microcontrolador Atmega328P en un circuito real e independiente de la placa de Arduino.

El bootloader es parte de la magia de Arduino. Es un pequeño código almacenado en su memoria que se ejecuta al arrancar y que se encarga de leer datos por su puerto serie y almacenarlo en la memoria flash. Es decir, que nos permite cargar nuestro propio código sin necesidad de usar un programador externo. Los Atmega328P se pueden adquirir a muy bajo precio, pero en la mayoría de los casos vendrán sin su correspondiente bootloader. Así que, si no lo trae, lo primero que tendremos que hacer es cargar el bootloader en el microcontrolador. La buena noticia es que podemos hacerlo con la ayuda de la propia placa Arduino. Esta es la lista de materiales que vamos a necesitar:

- Arduino UNO.
- Protoboard.
- Un Atmega328P (ojo, no un Atmega328, son distintos).
- Oscilador 16MHz (tipo HC49S)
- Dos condensadores cerámicos de 22pF.
- Dos resistencias (1K y 10K)
- Cables

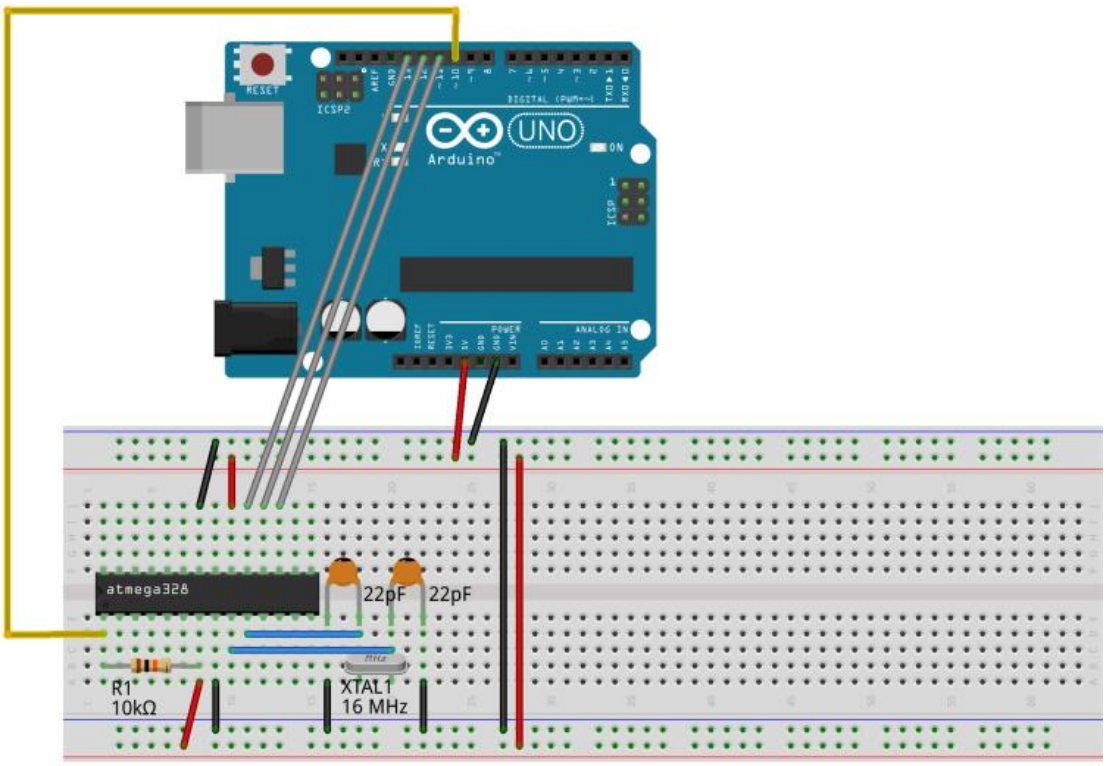


Distribución de pines del Atmega328P

Lo mínimo necesario para que el Atmega328P funcione es, por supuesto, conectar los pines 7 y 20 a VCC (5V) y el 8 y 22 a tierra. Aunque el Atmega328P puede funcionar con su reloj interno de 8MHz, sin bootloader no vamos a poder activarlo, así que usaremos uno externo a 16MHz que se conecta a las patillas

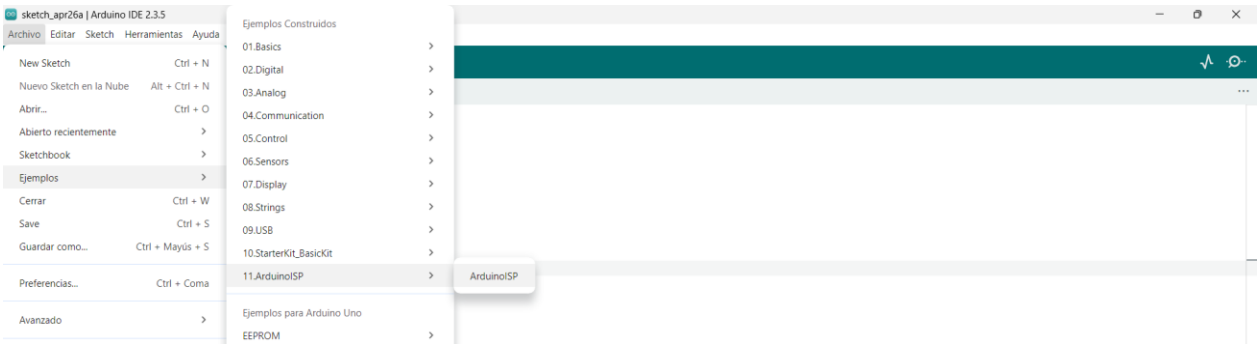
9 y 10, con sus correspondientes condensadores (he usado dos cerámicos de 22pF). Si tienes un resonador a 16MHz (yo no tenía ninguno a mano) puedes usarlo en lugar del oscilador y ahorrarte los dos condensadores.

El circuito quedaría como se muestra en la siguiente imagen:

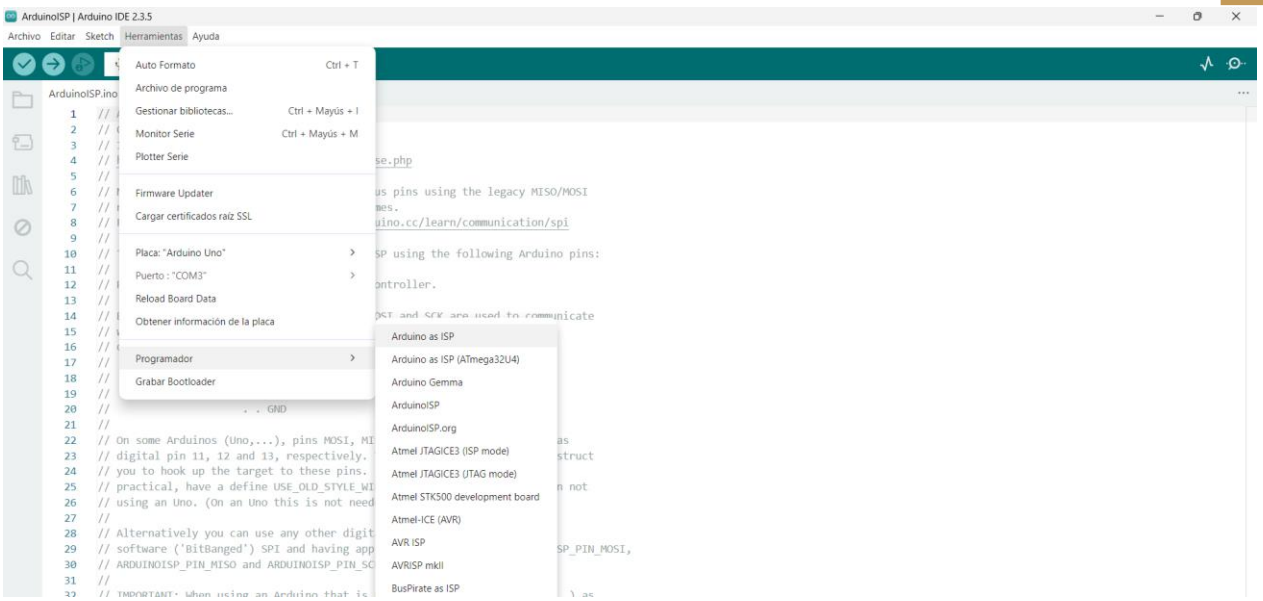


El programa que se encarga de escribir el bootloader necesita (como mínimo) una conexión al pin de Reset (con la resistencia de 10K) y a los pines 13, 12 y 11.

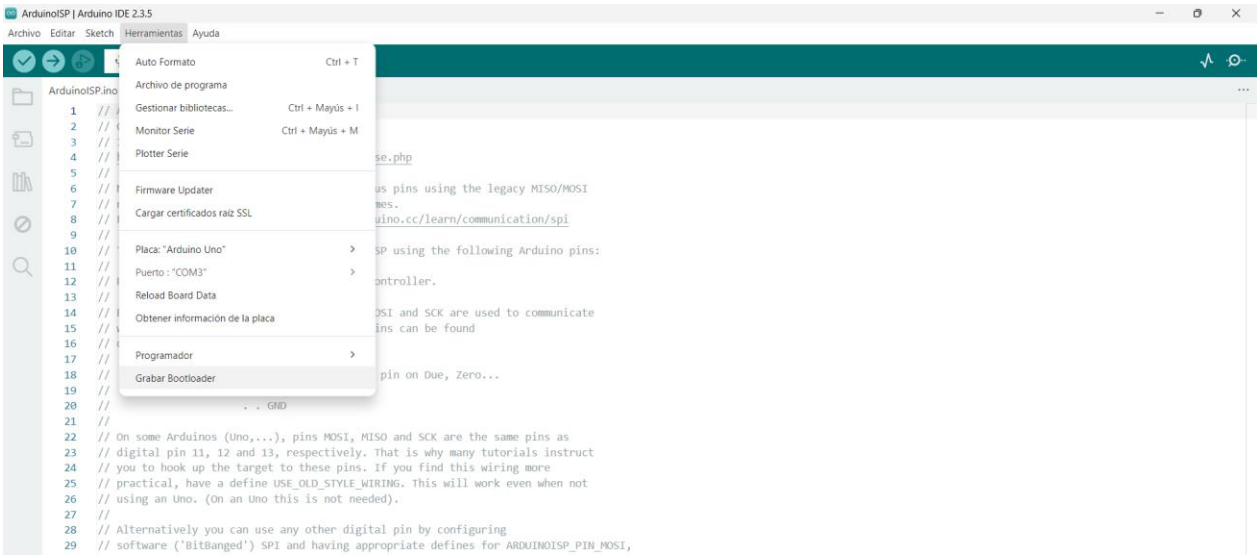
Ahora, con el Arduino conectado al PC y el entorno de desarrollo ejecutándose, seleccionamos en el menú Archivo: Ejemplos > Arduino ISP.



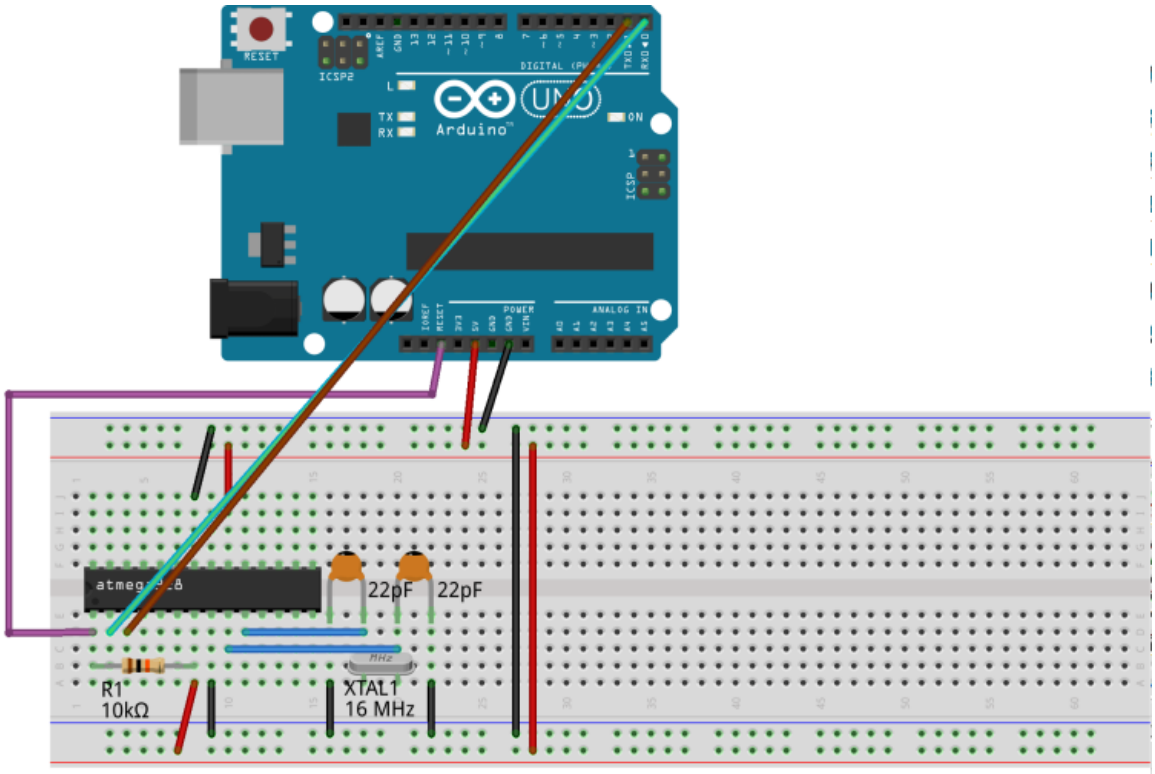
En el menú Herramientas: *Programador > Arduino as ISP.*



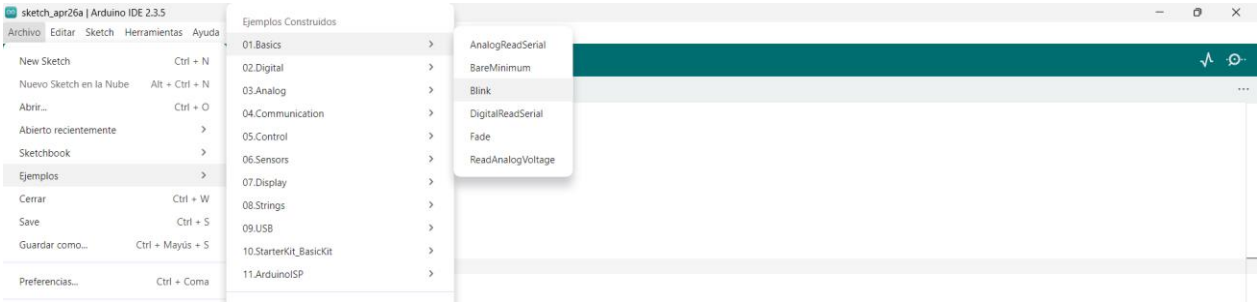
Y finalmente, también en el menú Herramientas, seleccionamos *Grabar Bootloader*.



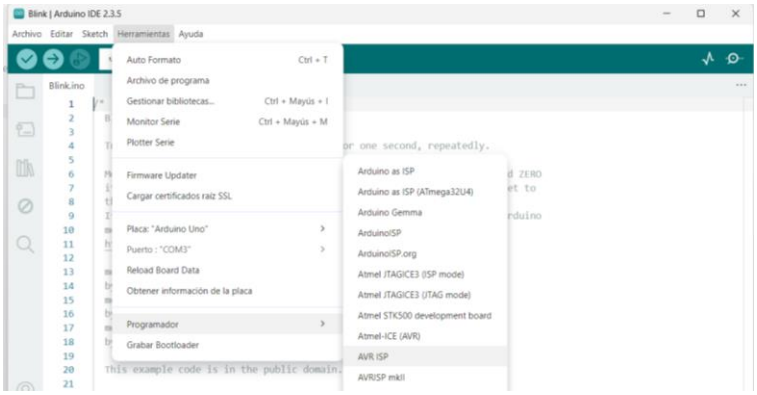
Sólo nos queda pulsar el botón para cargar el código en el microcontrolador (el botón de la flechita). Si todo va bien, nuestro Atmega328P ya tiene grabado su bootloader y está listo para que podamos cargar nuestro programa en él. Podríamos pincharlo en el zócalo de una placa Arduino y programarlo normalmente para después volver a ponerlo en la protoboard, pero la buena noticia es que podemos programarlo directamente en la protoboard conectando los pines TX y RX a la placa Arduino. Ojo, antes hay que quitar el Atmega328P que hay en la placa Arduino. Las conexiones quedarían así:



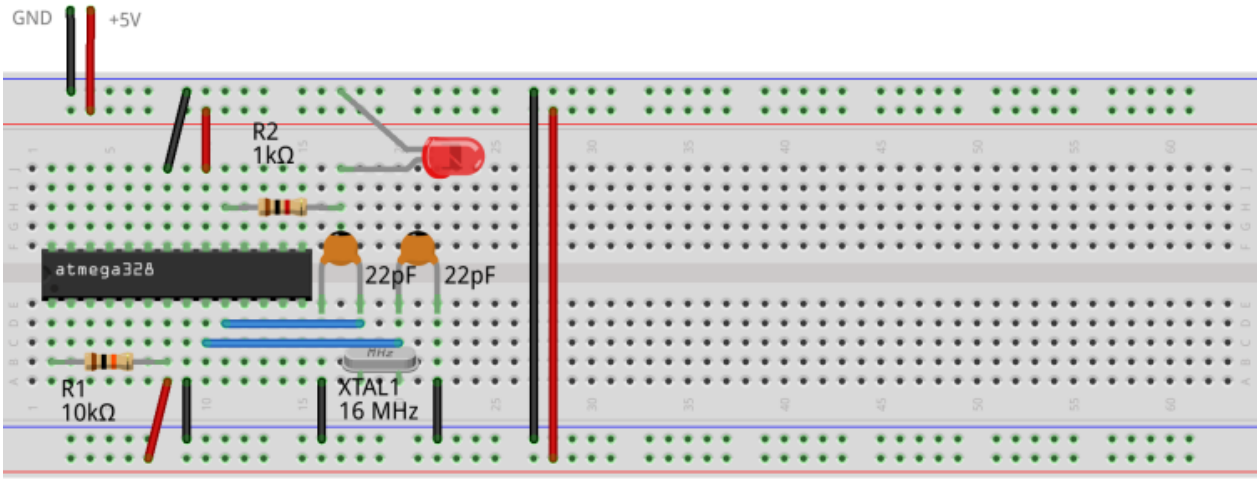
Ahora vamos a programarlo. Usaremos el famoso Blink que trae de ejemplo el entorno de desarrollo. Está en el menú Archivo > Ejemplos > Basic > Blink.



Cambiamos el programador en Herramientas > Programador > AVR ISP.

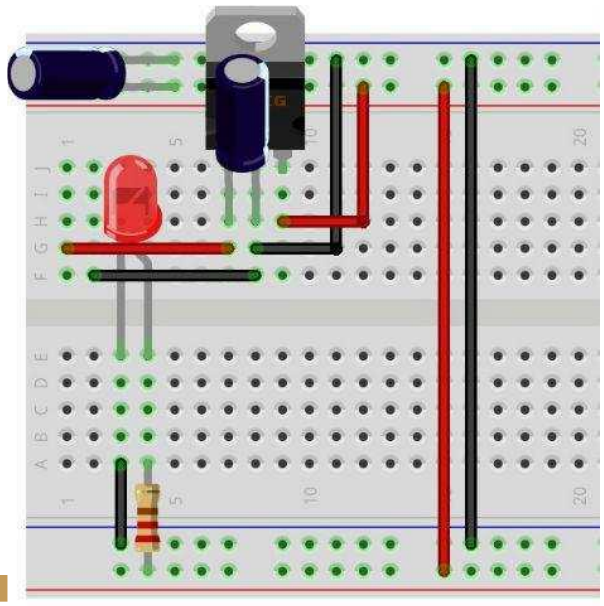


Finalmente pulsamos el botón de la flechita para cargar nuestro programa. Ahora el código está grabado en el microcontrolador y cada vez que le suministremos 5V y lo pongamos en marcha, lo ejecutará. El programa Blink hace parpadear un led de la placa Arduino que está conectado al pin 13, así que tendremos que poner un led y una resistencia de 1K en la protoboard para poder probar el montaje.



Nuestro circuito aún no estaría completo, ya que en un circuito real es necesario usar algún tipo de regulador de voltaje a 5V como el l7805cv, incluso aunque alimentemos el montaje con pilas, pero para simplificar, alimentaré el circuito con una fuente de alimentación estabilizada. Recuerda que hay que suministrar 5V (el circuito apenas consume 20 miliamperios). Si todo está correcto, el led comenzará a parpadear gracias a nuestro circuito basado en el Atmega328.

La idea aquí es proporcionarle a tu Atmega328 una alimentación regulada para que no necesites una fuente de exactamente 5V conectada al chip.



Podrías alimentar el Atmega328 a 5V simplemente con el regulador 7805. Sin embargo, se suelen añadir un par de condensadores de acoplo y desacoplo (los de 10µF) para que la salida del 7805 sea más estable. Además, si conectas un LED (con su respectiva resistencia), podrás ver de forma sencilla si estás alimentando tu Standalone correctamente.

Aunque puedes organizar el circuito como quieras, conviene que los condensadores de

acoplo y desacoplo estén lo más cerca posible del 7805.

Los dos cables que puedes ver a la izquierda (los que no están conectados a ningún elemento) son los que utilizarás para alimentar tu **Standalone.**

Lectura 11. Contadores

¿Qué son los contadores?

Un contador es un dispositivo que va acumulando el número de veces que se realiza una acción.

La implementación de esta tarea con microcontroladores se puede realizar de distintas maneras, todo depende de las necesidades o diseño del programador.

Aplicaciones de los contadores

Existen múltiples aplicaciones para el uso de los contadores, pero todas tienen algo en común, llevar el conteo de acciones que un microcontrolador detecta a través de pulsos eléctricos, a continuación, se muestran algunas de estas aplicaciones.

Contador binario

En este ejemplo vamos a usar el puerto B que ocupa los pines digitales del 8 al 13, el puerto B tiene un registro para configurar el sentido de transferencia, todos los puertos tienen uno y cuando se llama a la función **pinMode()** lo que esta función hace es tratar con estos registros de manera transparente al usuario, el DDRB (Data Direction Register for Port B) configura el sentido de transferencia de los pines del puerto B, si por ejemplo escribimos DDRB = 0xFF todo el puerto será salida, DDRB = 0x0F configura la parte alta del puerto como entrada y la parte baja como salida. Está claro que el bit que tiene un uno será configurado como salida y el bit con 0 será entrada.

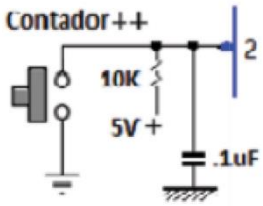
Un botón conectado en el pin 2 (Puerto D) será el encargado incrementar un contador que contará en binario generando una secuencia de números que va desde 0 a 15 (0000 a 1111) el estado del contador se pasará a la parte baja del puerto B (4 bits menos significativos) y se mostrará en cuatro LED's conectados a los pines 8,9,10 y 11.

DDRB = 0x0F

La instrucción DDRB = 0x0F (donde 0x0F es 15 en decimal, o 00001111 en binario) establece los bits del registro DDRB, que determina si un pin es entrada o salida.

En este caso, los 4 bits menos significativos (del bit 0 al bit 3) se establecen a 1, indicando que los pines correspondientes serán salidas.

El circuito electrónico para el botón contador es el siguiente, observe que el estado de "NO cuenta" es un nivel alto en el pin provisto por la resistencia de 10K, esto enmascara las interferencias y ruidos eléctricos que podrían afectar el contador. Para contar se debe llevar el pin a nivel bajo mediante el botón.



El programa para el contador binario quedaría de la siguiente manera:

```
/************************************************
* Contador binario, muestra el estado de cuenta en 4 bit's por la parte * baja del puerto B.
* El contador se incrementa en 1 cada vez que se oprime un botón colocado * en el pin 2 configurado como entrada.
* Placa Arduino: UNO R3
* Arduino IDE: 1.8.12
* www.firtec.com.ar
************************************************/
byte contador = 0;
unsigned char boton = 2; // Pin del botón
unsigned char estado = 0; // Estado del botón
unsigned char bandera = 0; // Bandera para contar
void setup() {
  pinMode(boton,INPUT_PULLUP);   //Pulsador para contar
  // -- Configuración de los pines para los LED's
  DDRB = 0x0F; // Parte baja del puerto es salida
  PORTB = 0; // El puerto es puesto a cero
}
void loop() {
  estado = digitalRead(boton);  // Mira el estado del botón
  if (estado == LOW && bandera== 0) {
     contador++;
     if(contador > 15){
        contador = 0;
     }
     PORTB = contador; // Coloca el binario por el puerto
     bandera = 1;
  }
  estado = digitalRead(boton);  // Mira el estado del botón
  if (estado == HIGH) {  // Si el nivel es alto el botón se soltó y se
  bandera = 0;          // prepara para contar otro número
  }
}
/********* Fin de Programa FIRTEC ARGENTINA ******/
```

Práctica 5. Contador en display de 7 segmentos

Instrucciones: Lee con atención lo que se indica en la siguiente práctica, cumpliendo con cada uno de los puntos que se especifican.

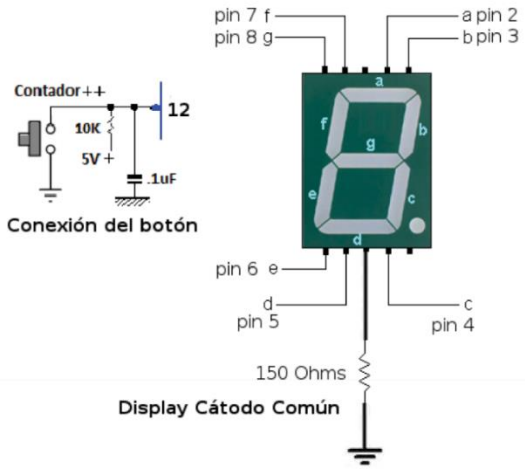
Objetivos:

- Colocar un botón pulsador al pin 12 y desde el pin 2 al 8 se conectan los siete segmentos de un display cátodo común.

- Considerando que los segmentos son simplemente diodos LED's conectados de manera conveniente en el formato de un dígito, lo único que tenemos que hacer en "enseñarle" a Arduino como encender estos LED's para mostrar números.

- Vamos a crear una matriz de array para formar en cada fila el número que vamos a ver.

Esquema de conexión:



Lista de materiales:

- Placa Arduino.
- Protoboard.
- Cable USB.
- Una resistencia de 10 KOhms, o el valor más aproximado.
- Una resistencia de 150 Ohms para el dsiplay.
- Un display de siete segmentos cátodo común.
- Un capacitor de 0.1 uF.
- Un botón, o un pulsador.
- Cable conexión.

Código de la práctica.

```
/*****************************************************************
* Contador de un dígito con display cátodo común.
* Se configura un pin como entrada (Pin12) donde se coloca un botón que * lleva el pin a un estado bajo para incrementar el contador. El estado * alto (contrario al estado de cuenta) se establece con una resistencia * de 10K conectado a +5V.
*
* Los segmentos están conectados de la siguiente forma:
* Pin 2 segmento a
* Pin 3 segmento b
* Pin 4 segmento c
* Pin 5 segmento d
* Pin 6 segmento e
* Pin 7 segmento f
* Pin 8 segmento g
*
* Placa Arduino: UNO
* Arduino IDE: 1.8.5
*
* www.firtec.com.ar
*****************************************************
unsigned char boton = 12; // Pin del botón
unsigned char estado = 0; // Estado del botón
const unsigned char num_array[10][7] =
{{ 1,1,1,1,1,1,0 }, // 0
{ 0,1,1,0,0,0,0 }, // 1
{ 1,1,0,1,1,0,1 }, // 2 // Codifica los números y segmentos
{ 1,1,1,1,0,0,1 }, // 3
{ 0,1,1,0,0,1,1 }, // 4
{ 1,0,1,1,0,1,1 }, // 5
{ 1,0,1,1,1,1,1 }, // 6
{ 1,1,1,0,0,0,0 }, // 7
{ 1,1,1,1,1,1,1 }, // 8
{ 1,1,1,0,0,1,1 }}; // 9
void Muestra_Dato(unsigned char); // Se declaran la función para mostrar datos
unsigned int retardo =0;
unsigned char contador=0,bandera=0; // Variables del programa
void setup(){
  pinMode(boton, INPUT); // Pin del botón es entrada
  pinMode(13, OUTPUT); // Pin del LED de la placa
```

Trabajar con array

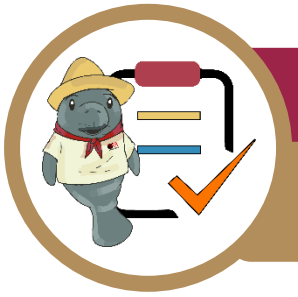
Para trabajar con el display se ha definido una cadena de bytes de dos dimensiones llamada num_array, esta cadena tiene 10 filas por 7 columnas y básicamente es una matriz. Como solo vamos a mostrar números de 0 a 9 nuestra matriz tendrá 10 filas y como son 7 segmentos, la matriz tendrá 7 columnas, una por cada segmento a encender o apagar. Cada una de esas columnas es un LED encendido o apagado dependiendo de si es "1" o "0", cuando se oprima el botón, el microcontrolador coloca el primer segmento en los pines de Arduino y solo encienden los LED's que forman el numero 1, posteriormente el 2, 3, 4, etc.

```
  digitalWrite(13,LOW ); // LED apagado
    // -- Configuración de los pines para los segmentos
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  Muestra_Dato(contador);
}
void loop(){
  estado = digitalRead(boton); // Mira el estado del botón
  if (estado == LOW && bandera== 0) {
    contador++;
    if(contador == 10)
      contador =0;
     Muestra_Dato(contador);
     bandera = 1;
  }
  estado = digitalRead(boton); // Mira el estado del botón
  if (estado == HIGH)
    bandera = 0;
}
void Muestra_Dato(unsigned char dato){
  unsigned char pin = 2;
  for (unsigned char j=0; j < 7; j++) {
  digitalWrite(pin, num_array[dato][j]);
  pin++;
  }
}
/*************** Fin de Programa FIRTEC ARGENTINA ******************/
```

Copia este código al IDE de Arduino y súbelo a la placa, conecta todos los componentes de acuerdo al diagrama proporcionado anteriormente y contesta las siguientes preguntas:

- ¿Qué tipos de datos se utilizaron en el código?
- ¿Cuáles fueron las instrucciones utilizadas en el código?
- ¿Qué tipo de entradas y salidas se utilizaron y cuáles fueron los componentes que se les asignaron?
- ¿Cuál fue el motivo por el que se utilizó un array?

Realiza un reporte de la práctica debidamente estructurado y entrega al docente.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 6

Práctica 5: Contador en display de 7 segmentos

DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Reporte de práctica		**Fecha**	
Formación laboral básica: Robótica **Submódulo 1:** Microcontroladores y microprocesadores		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Conecta correctamente el display de 7 segmentos y el botón de acuerdo con el esquema			2		
2.	Copia y carga correctamente el código al IDE de Arduino sin errores			1		
3.	El display muestra los números del 0 al 9 correctamente al presionar el botón			2		
4.	Describe correctamente los tipos de datos utilizados en el código (ej. unsigned char, int)			1		
5	Identifica adecuadamente las instrucciones usadas en el código (pinMode, digitalWrite, if, etc.)			1		
6	Explica correctamente qué tipo de entradas y salidas se usaron y qué componentes controlan			1		
7	Justifica de forma clara el uso del array num_array en el contexto del display			1		
8	Entrega un reporte completo, estructurado y claro de la práctica			1		
				CALIFICACIÓN		

Lectura 12. Interrupciones

En ocasiones, durante la realización de algunas prácticas, nos hemos podido dar cuenta de que Arduino puede estar perdiendo información. Esto es debido a que el microcontrolador está ocupado realizando el cometido que le hemos programado.

Pongamos el ejemplo más sencillo que podamos encontrar, por ejemplo, al utilizar delay's.

Al utilizar un **delay**, Arduino se para durante el tiempo que le ordenamos, en consecuencia, ¿qué sucede si durante ese período de tiempo un sensor detecta alguna entrada de datos, o un botón es activado?, pues no ocurre nada, Arduino seguirá con su **delay** obviando la entrada de datos que ha ocurrido precisamente en ese lapso de tiempo . . .

Para solventar este problema, lo recomendable es utilizar interrupciones.

Una interrupción la podemos definir como una llamada al microcontrolador, el cual, deja lo que está ejecutando y atiende dicha llamada.

Esta llamada o interrupción, normalmente, "lleva" al microcontrolador a otra parte del código que debe ejecutarse con mayor prioridad. Una vez ejecutado ese trozo de código, el microcontrolador vuelve al punto anterior, es decir, a la línea de la instrucción donde lo había dejado antes de recibir la interrupción.

Arduino UNO posee dos pines para crear interrupciones. En este caso, serán interrupciones externas, ya que las vamos a generar nosotros desde fuera del microcontrolador.

Estas interrupciones son: INT0, INT1, las cuales están vinculadas con los pines 2 y 3 respectivamente.

A este tipo de interrupciones también las podemos llamar interrupciones hardware.

La función attachInterrupt

Para desarrollar la interrupción con Arduino, tenemos una función que genera la interrupción deseada.

La función **attachInterrupt** tiene la siguiente sintaxis:

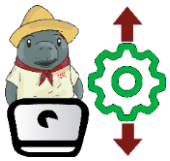
attachInterrupt (n°int, nombre función, modo)

Veamos cada uno de los parámetros de la función:

- N° int: número de la interrupción. Si utilizamos la interrupción 0, el pin a utilizar será el 2. Si utilizamos la interrupción 1, el pin será el 3.
- Nombre función: nombre que le damos a la función interrupción que deseamos llamar.

- Modo: Define cuando la interrupción deberá ser activada. Observamos 4 formas de activación:

- CHANGE: Se activa cuando el valor en el pin pasa de HIGH a LOW o de LOW a HIGH. Es el modo normalmente empleado.
    - LOW: Se activa cuando el valor en el pin es LOW.
    - RISING: Se activa cuando el valor del pin pasa de LOW a HIGH.
    - FALLING: Se activa cuando el valor del pin pasa de HIGH a LOW.



Práctica 6. Interrupciones mediante un botón

Instrucciones: Lee con atención lo que se indica en la siguiente práctica, cumpliendo con cada uno de los puntos que se especifican.

Objetivos:

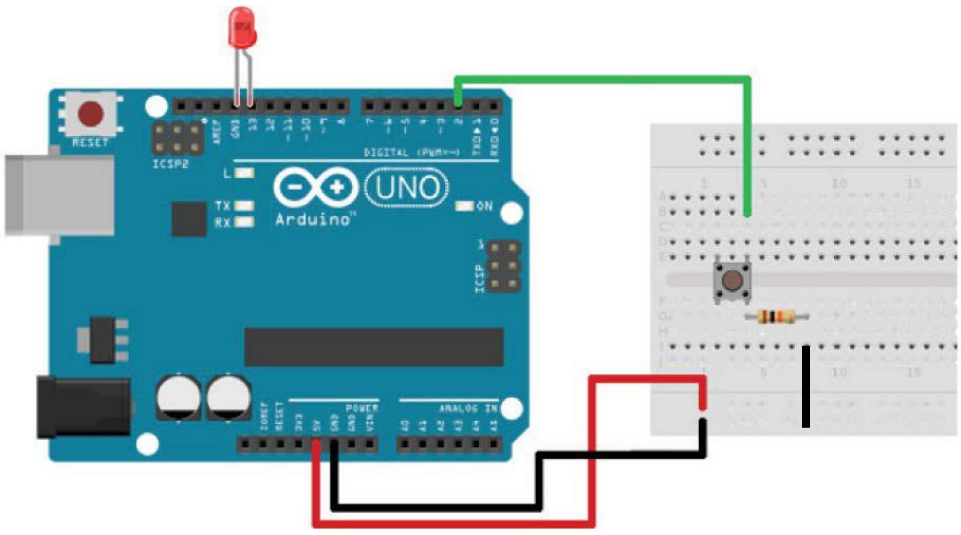
- En esta práctica se desea crear una interrupción mediante hardware a través de un botón. Dicho botón encenderá un diodo led.

- Para demostrar la utilidad y efectividad de la interrupción, primero se llevará a cabo el mismo esquema sin interrupción, aplicándole además, un bucle de retardo mediante un **for** o un **delay** para acentuar el posible retraso en el encendido de dicho led.

- Después se comprobará la efectividad de la interrupción añadiéndola al programa.

Esquema de conexión:

Como se ha comentado anteriormente, para poder estudiar bien el funcionamiento de una interrupción normalmente se suele utilizar el montaje de encendido de un led mediante un botón.
Por tanto, el esquema de conexión para esta práctica será el mismo que se utiliza para la práctica del botón. Aquí se ha considerado escoger la interrupción INT1, es decir, la interrupción asociada al pin 2 de Arduino.



Lista de materiales:

- Placa Arduino
- Protoboard
- Cable USB
- 1 botón
- 1 resistencia de 220 o 330 Ohms
- 1 led
- Cables conexión

Código de la práctica.

Antes de empezar a experimentar con las interrupciones, veamos un código donde se introduce un bucle de retardo para simular la pérdida de información cuando Arduino está ocupado. Aquí, el botón no responderá a nuestra petición de encendido con solo pulsar, y se deberá insistir para que nos obedezca.

Código de encendido de un led sin interrupción.

```
/* Retardo del encendido de un led mediante botón */
int boton = 7;
int led=13;
int vb = 0;  //variable que almacena el valor del botón

void setup () {
 pinMode (boton, INPUT);
 pinMode (led, OUTPUT);
 Serial.begin (9600);
}

void loop() {
 for (int t=0; t<500;t++) {  //bucle de retardo
 Serial.println (t);
 }
  vb = digitalRead (boton);
  if (vb == HIGH) {
    digitalWrite (led, HIGH);
    delay (220) ;
  }
  else {
    digitalWrite (led, LOW);
    delay (220);
  }
}
```

Copia este código al IDE de Arduino y súbelo a la placa, conecta todos los componentes de acuerdo al diagrama proporcionado anteriormente y contesta la siguiente pregunta:

- ¿El programa responde a la brevedad y enciende el led después de haber pulsado el botón?

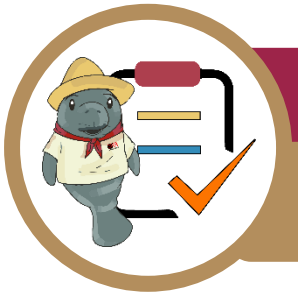
Ahora veamos el código donde se emplea una interrupción:

```
/* Encendido de un led mediante una interrupción */
int boton = 2;
int led=13;
int vb = 0;  //variable que almacena el valor del botón

void setup () {
pinMode (boton, INPUT);
pinMode (led, OUTPUT);
Serial.begin (9600);
attachInterrupt(0,parpadeo, HIGH);
}

void loop () {
for (int t=0;t<500;t++) {
Serial.println (t);
}
}

void parpadeo(){
vb = digitalRead (boton);
  if (vb == HIGH) {
    digitalWrite (led, HIGH);
    delay (220) ;
  }
  else {
    digitalWrite (led, LOW);
    delay (220);
 }
}
```

Ahora copia este código y contesta las siguientes preguntas:

- ¿El programa responde a la brevedad y enciende el led después de haber pulsado el botón?
- ¿A qué crees que se deba? (Argumenta muy bien tu respuesta)
- ¿Consideras a las interrupciones un elemento primordial en la programación de microcontroladores? (Argumenta tu respuesta)

Realiza un reporte de la práctica debidamente estructurado y entrega al docente.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 7

Práctica 6: Interrupciones mediante un botón

DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Reporte de práctica		**Fecha**	
Formación laboral básica: Robótica **Submódulo 1:** Microcontroladores y microprocesadores		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Conectó correctamente el circuito (Arduino, botón, resistencia y LED).			2		
2.	Cargó y ejecutó correctamente el código sin interrupciones en Arduino.			1		
3.	Contestó de manera clara si el LED respondió al presionar el botón sin interrupciones.			1		
4.	Cargó y ejecutó correctamente el código con interrupciones en Arduino.			2		
5	Respondió si el LED reaccionó adecuadamente con interrupciones.			1		
6	Argumentó de forma clara y lógica la diferencia en el funcionamiento con interrupciones.			2		
7	Presentó el reporte estructurado (introducción, desarrollo, conclusiones, limpieza y orden).			1		
				CALIFICACIÓN		

Lectura 13. Temporizadores

Un timer (temporizador) en Arduino UNO es un componente del microcontrolador (ATmega328P) que permite realizar tareas cronometradas automáticamente sin necesidad de ocupar el ciclo principal (loop()). Esto es útil para ejecutar funciones cada cierto tiempo, generar señales PWM, medir tiempos o manejar interrupciones por tiempo.

Un timer es un contador que incrementa automáticamente con cada "tick" del reloj del microcontrolador. Puedes configurar este contador para que:

- Genere una interrupción después de cierto tiempo.
- Reinicie o continúe contando.
- Ejecutes una función automáticamente sin usar delay().

Esto es muy útil cuando necesitas hacer varias cosas a la vez, como:

- Leer sensores continuamente.
- Parpadear LEDs sin bloquear el loop principal.
- Ejecutar tareas periódicas

El Arduino UNO (basado en el chip ATmega328P) tiene 3 timers:

Timer	Tamaño	Usado por funciones estandar
Timer0	8 bits	delay(), millis(), micros()
Timer1	16 bits	Servo, TimerOne (más preciso)
Timer2	8 bits	tone(), MsTimer2, PWM extra

¿Cómo se usan los timers?

- *Con librerías*

Una librería es un conjunto de funciones y definiciones reutilizables que facilitan la programación al encapsular tareas complejas o repetitivas. Permiten al programador usar funcionalidades avanzadas sin tener que escribir todo el código desde cero.

Usar librerías para timers permite:

- Ejecutar tareas cada cierto tiempo sin usar **delay()**.
- Mantener el código simple y legible.
- Evitar errores al manipular registros directamente.
- Hacer código más portable y mantenible.

Las librerías más comunes para timers son:

Librería	Timer que usa	Resolución	Ventajas principales
TimerOne	Timer1 (16-bit)	Alta	Precisión, fácil de usar
MsTimer2	Timer2 (8-bit)	Media	Ideal para tareas cortas (ms)
TimerThree	Timer3 (otros Arduinos)	Alta	Para placas Mega y similares

En el caso del Arduino UNO, las más útiles son TimerOne y MsTimer2.

Ejemplo detallado con TimerOne (la más precisa)

1. Instalación de la librería:

   Si no la tienes instalada:

   - Abre el IDE de Arduino.
   - Ve a "Sketch > Include Library > Manage Libraries..."
   - Busca TimerOne e instálala.

2. Código de ejemplo: Parpadear un LED cada segundo

```
#include <TimerOne.h>  // Incluir la librería

void setup() {
  pinMode(13, OUTPUT);  // Pin del LED
  Timer1.initialize(1000000);  // 1,000,000 microsegundos = 1 segundo
  Timer1.attachInterrupt(parpadeo);  // Ejecuta "parpadeo" cada segundo
}

void loop() {
  // Aquí puedes poner cualquier otra tarea sin bloquear
}

void parpadeo() {
  digitalWrite(13,!digitalRead(13));  // Cambia el estado del LED
}
```

3. Explicación paso a paso

   #include <TimerOne.h>

   Importa la librería TimerOne que simplifica el uso del Timer1.

   Timer1.initialize(1000000);

   Inicializa el Timer1 con un periodo de 1,000,000 microsegundos, es decir, 1 segundo.

Timer1.attachInterrupt(parpadeo);

Indica que se ejecutará la función parpadeo() automáticamente cada 1 segundo sin necesidad de usar delay().

parpadeo()

Función que se ejecuta cada vez que el timer llega al tiempo indicado. En este caso, cambia el estado del LED.

► *Sin librerías*

Cuando usamos un timer sin librerías, accedemos a registros internos del microcontrolador ATmega328P (el cerebro del Arduino UNO). Estos registros controlan cómo funciona el contador interno (timer) y nos permiten hacer cosas como:

- Contar hasta un número específico.
- Ejecutar una interrupción automática cuando llega a ese número.
- Repetir el ciclo (modo CTC o PWM).
- Controlar la frecuencia con precisión.

¿Qué es el Timer1?

- Es un timer de 16 bits (puede contar hasta 65,535).
- Tiene más precisión y rango que Timer0 o Timer2 (que son de 8 bits).
- Muy útil para generar interrupciones temporizadas o PWM de alta resolución.

Código de ejemplo: Parpadear un LED cada segundo

```
void setup() {
  pinMode(13, OUTPUT);
  noInterrupts(); // Desactiva interrupciones mientras configuramos el timer
  TCCR1A = 0;     // Registro de control A - limpio
  TCCR1B = 0;     // Registro de control B - limpio
  TCNT1  = 0;     // Reinicia el contador a 0
  OCR1A = 15624;  // Valor al que el timer contará antes de reiniciarse (modo CTC)
            // (16 MHz) / (1024 prescaler) / (1 Hz) - 1 = 15624
  TCCR1B |= (1 << WGM12); // Modo CTC (Clear Timer on Compare Match)
  TCCR1B |= (1 << CS12) | (1 << CS10); // Prescaler de 1024
  TIMSK1 |= (1 << OCIE1A); // Habilita interrupción por comparación A
  interrupts(); // Activa interrupciones de nuevo
}
ISR(TIMER1_COMPA_vect) {
  digitalWrite(13, !digitalRead(13));// Cambia el estado del LED cada 1 segundo
}
void loop() {
  // Código principal. No se bloquea gracias al timer.
}
```

Explicación del código:

- **pinMode(13, OUTPUT);**
  Configura el pin 13 como salida para encender o apagar un LED.

Configuración del timer:

- **noInterrupts();**
  Desactiva todas las interrupciones del microcontrolador mientras configuramos los registros, para evitar problemas.
- **TCCR1A = 0; y TCCR1B = 0;**
  Limpia los registros de control del Timer1. Queremos configurar todo desde cero.
- **TCNT1 = 0;**
  Reinicia el valor del contador del Timer1.
- **OCR1A = 15624;**
  Este es el valor tope al que el timer cuenta antes de lanzar una interrupción.
  - Arduino UNO tiene un reloj de 16 MHz.
  - Queremos una interrupción cada 1 segundo.
  - Con un prescaler de 1024, el cálculo es:
    OCR1A = (16,000,000 / 1024) / 1 Hz - 1 = 15,624
    Esto hace que la interrupción ocurra cada 1 segundo.

Modos del timer

- **TCCR1B |= (1 << WGM12);**
  Activa el modo CTC (Clear Timer on Compare). Esto reinicia el contador a 0 automáticamente cuando llega al valor de OCR1A.
- **TCCR1B |= (1 << CS12) | (1 << CS10);**
  Establece el prescaler a 1024. Esto divide la frecuencia del reloj, para que el timer cuente más lentamente.

Activación de interrupciones

- **TIMSK1 |= (1 << OCIE1A);**
  Habilita la interrupción por comparación A, es decir, cuando el valor del contador llegue a OCR1A, se llamará a una función especial llamada ISR.

ISR — Rutina de Servicio de Interrupción

- **ISR(TIMER1_COMPA_vect)**
- Esta función especial se ejecuta automáticamente cuando ocurre la interrupción.

- **digitalWrite(13, !digitalRead(13));**
  Esto alterna el estado del pin 13 (enciende y apaga el LED) cada vez que ocurre la interrupción (cada segundo en este caso).

Ventajas de este enfoque

- Precisión alta (ideal para tareas críticas).
- No bloqueas el loop().
- Controlas los recursos directamente (sin depender de funciones ocultas).
- Puedes usar más de un timer a la vez.

Advertencias

- Si usas Timer1, no puedes usar la librería Servo, ya que también lo utiliza.
- Si programas mal un registro, puedes bloquear temporizadores del sistema, lo que hace que delay(), millis() o micros() dejen de funcionar.
- Este método requiere conocer bien el hardware.

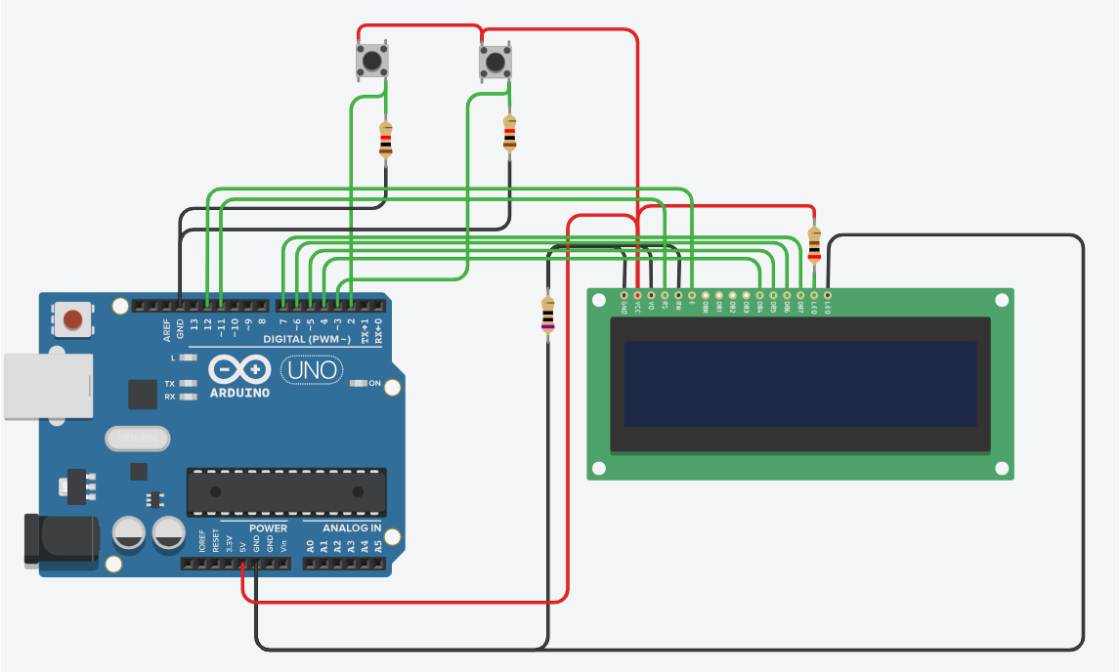
Práctica 7. Reloj

Instrucciones: Lee con atención lo que se indica en la siguiente práctica, cumpliendo con cada uno de los puntos que se especifican.

Objetivos:

- Configurar registros de temporizadores para la creación de un reloj.
- Conectar dos botones para configurar la hora exacta.

Esquema de conexión:



Lista de materiales:

- Placa Arduino
- Protoboard
- Cable USB
- 2 botones
- 2 resistencias de 1 KOhms
- 1 resistencia de 200 Ohms o aproximado
- 1 resistencia de 10 Ohms o aproximado
- Cables conexión
- 1 pantalla LCD 16x2

Código de la práctica.

```
#include<LiquidCrystal.h>

const byte RS = 11 , EN = 12;
const byte DB4 = 4 , DB5 = 5 , DB6 = 6 , DB7 = 7 ;
LiquidCrystal lcd (RS, EN, DB4, DB5, DB6, DB7) ;

int contor = 0 ;

int secunde = 0 ;
int minute = 0 ;
int ore = 0 ;

void setup ( )
{
        lcd.begin (16 , 2 ) ;
        lcd.print ( "Hora exacta : " ) ;
        DDRD |= (1 << 2 ) | (1 << 3 ) ;
        EICRA |=(1<<ISC11 ) | (1<< ISC10 ) | (1 << ISC01 ) | (1 << ISC00 ) ;
        EIMSK |=(1 << INT1 ) | (1 << INT0 ) ;
        EIFR |=(0 << INTF1 ) | (0 << INTF0 ) ;
        PCICR |=(0 << PCIE2 ) | (0 << PCIE1 ) | (0 << PCIE0 ) ;

        TCCR0A = (1 << WGM01) | (0 << WGM00) ;
        OCR0A = 0xF9 ;
        TIMSK0 |= (1 << OCIE0A) ;
        TCCR0B = (1 << CS02 ) | (0 << CS01 ) | (0 << CS00 ) ;
        SREG |= (1 << SREG_I ) ;
}

void loop ( )
{
        lcd.setCursor ( 0 , 1 ) ;
        if( ore < 10 )
        {
                lcd.print ( 0 ) ;
                lcd.setCursor ( 1 , 1 ) ;
        }
        lcd.print(ore) ;
        lcd.setCursor( 2 , 1 ) ;
```

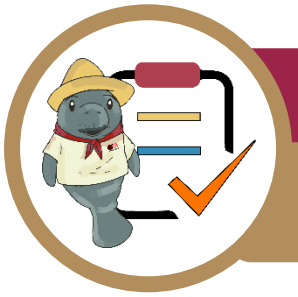
```
        lcd.print( " : " ) ;
        lcd.setCursor( 3 , 1 ) ;
        if( minute < 10 )
        {
                lcd.print(0);
                lcd.setCursor(4,1) ;
        }
        lcd.print( minute ) ;
        lcd.setCursor ( 5 , 1 ) ;
        lcd.print( " : " ) ;
        lcd.setCursor( 6 , 1 ) ;
        if( secunde < 10 )
        {
                lcd.print ( 0 ) ;
                lcd.setCursor ( 7 , 1 ) ;
        }
        lcd.print( secunde ) ;
}
inline void incrementOre()
{
        ++ore ;
        if(ore >= 24 )
        {
        ore %= 24;
        }
}
inline void incrementMinute ( )
{
        ++minute ;
        if (minute >= 60 )
        {
                incrementOre () ;
                minute %= 60;
        }
}
ISR (TIMER0_COMPA_vect)
{
        SREG &= ~(1 << SREG_I);
        ++contor;
        if( contor >= 250 )
        {
```

```
                ++secunde ;
                contor = 0 ;
                if ( secunde >= 60 )
                {
                incrementMinute() ;
                secunde %= 60;
                }
        }
        SREG |= (1 << SREG_I ) ;
}
ISR ( INT0_vect )
{
        SREG &= ~(1 << SREG_I);
        incrementMinute ( ) ;
        SREG |= (1 << SREG_I ) ;
}

ISR ( INT1_vect )
{
        SREG &= ~(1 << SREG_I);
        incrementOre ( ) ;
        SREG |= (1 << SREG_I) ;
}
```

Copia este código al IDE de Arduino y súbelo a la placa, conecta todos los componentes de acuerdo al diagrama proporcionado anteriormente y contesta las siguientes preguntas:

- ¿Cuáles instrucciones para configurar los registros de temporizadores identificas en el programa?
- ¿Para qué sirve la librería que se incluye al inicio del código?
- ¿Consideras a los temporizadores un elemento primordial en la programación de microcontroladores? (Argumenta tu respuesta)

Realiza un reporte de la práctica debidamente estructurado y entrega al docente.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 8

Práctica 7: Reloj

DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s);**	
Producto: Reporte de práctica		**Fecha:**	
Formación laboral básica: Robótica **Submódulo 1:** Microcontroladores y microprocesadores		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente:**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Conecta correctamente todos los componentes (LCD, botones, resistencias) según el esquema.			2		
2.	Copia y carga correctamente el código al IDE de Arduino sin errores.			1		
3.	El display muestra correctamente la hora actualizada en formato HH:MM:SS.			2		
4.	Identifica correctamente las instrucciones utilizadas para configurar los registros de temporizador.			1		
5	Explica correctamente la función de la librería LiquidCrystal.h			1		
6	Argumenta si los temporizadores son primordiales en la programación de microcontroladores.			1		
7	Reconoce y explica el uso de interrupciones externas (INT0, INT1) y su función dentro del programa.			1		
8	Entrega un reporte completo, claro y bien estructurado de la práctica			1		
				CALIFICACIÓN		

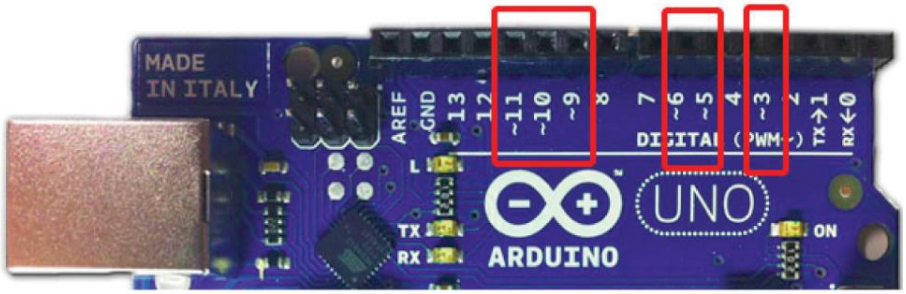
Lectura 14. PWM (Modulación por ancho de pulso)

Las siglas PWM provienen de la palabra en inglés Pulse Wide Modulation, modulación por amplitud de pulsos.

Es una forma de «simular» una señal analógica partiendo de una señal digital. Se envían 5v y 0v (1's y 0's) con cierta duración, simulando así una señal analógica.

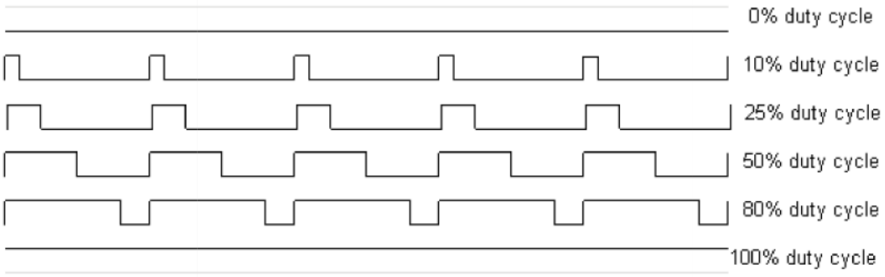
Para utilizar la señal PWM, Arduino dispone de unos pines especiales: el pin 3, 5, 6, 9, 10 y 11. Estos pines llevan serigrafiada en la placa la señal o el símbolo «~».

Los pines asignados a PWM también se pueden usar como puertos de uso general, tienen todas las características de cualquier pin de puerto.



Para gestionar este tipo de señal deberemos ajustar los valores entre 0 y 255, que corresponden a 0v y 5v, respectivamente. De esta forma, la señal PWM generará, según lo que le especifiquemos, una señal cuadrada o pulso cuadrado con una duración y frecuencia establecidas. Se muestra una imagen para poder observar lo que se denomina duty cycle, o ciclo de actividad, es decir, qué anchura tiene el pulso.

El ciclo de trabajo de una señal periódica es el ancho relativo de su parte positiva en relación con el período, el duty cycle es el tiempo que la salida está a uno o a un nivel alto.



La modulación por ancho de pulsos también es una técnica en la que se modifica el ciclo de trabajo de una señal periódica, ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

En Arduino la frecuencia de PWM es de 500Hz. Pero es un valor que puede modificar en caso de que lo necesitemos.

El control de potencia por modulación del ancho de pulso es ampliamente usado para el control de motores, iluminación, etc. En la actualidad existen muchos circuitos integrados en los que se implementa la modulación PWM, por ejemplo, para lograr circuitos funcionales que puedan controlar fuentes conmutadas, controles de motores, controles de elementos termoeléctricos, choppers para sensores en ambientes ruidosos, aplicaciones robóticas, etc.

Los microcontroladores usan varios modos de PWM, uno de ellos el Fast PWM que puede ser generado 8 (256), 9 (512) y 10 (1024) bits, una resolución mayor de 8 bits solo es posible usando un timer de 16 bits.

*Recordando **analogWrite()***

Para poder gestionar los pines de la señal PWM empleamos la función **AnalogWrite (pin, valor)**, explicada en páginas anteriores.

Esta función nos permite enviar los valores que deseamos para generar una señal PWM.

Por ejemplo, si introducimos analogWrite (11, 255), la salida por el pin 11 sería una señal cuadrada con su máximo valor de trabajo, es decir 5 v. Si los valores fuesen analogWrite (11, 128), se genera una señal cuadrada con un valor de la mitad que la anterior, etc.

Por tanto, la variable <<valor>> determinará el duty cicle anteriormente comentado.

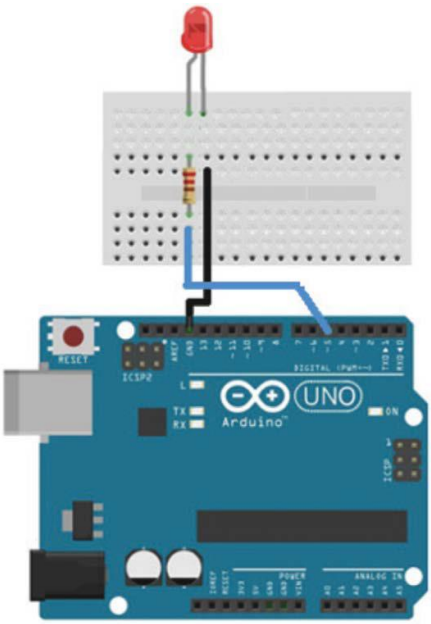
Práctica 8. Luminosidad variable

Instrucciones: Lee con atención lo que se indica en la siguiente práctica, cumpliendo con cada uno de los puntos que se especifican.

Objetivos:

- En esta práctica se deberá realizar un circuito en el que un led cambie la intensidad de su brillo.
- Para conseguir este efecto hay que recordar que se deberá utilizar la función analogWrite (pin, valor) y variar el duty cicle, obteniendo así un efecto de señal analógica y, por tanto, de brillo en el led.
- Se puede ir aumentando el brillo y, una vez llegado al límite (valor 255), bajar hasta su apagado (valor 0). Una vez llegado a 0, deberá volver a comenzar y aumentar poco a poco el brillo.

Esquema de conexión:



Lista de materiales:

- Placa Arduino
- Protoboard
- Cable USB
- Una resistencia de 220 Ohms
- Un led
- Cable conexión

Código de la práctica.

```
/*Luminosidad variable de un led mediante pulsos PWM*/
void setup(){
 pinMode (5, OUTPUT);
}

void loop(){
 analogWrite (5,LOW);
 brillo();
}

void brillo(){
 for (int i=O; i<=255; i=i+lO) {
 analogWrite (5,i);
 delay (200);
 }
}
```

Variando el entero (en este caso, 10), podemos obtener un efecto interesante de apagado y encendido de un led.

En este código se ha creado una función llamada brillo ().

Expliquemos varias líneas de este código.

pinMode (5, OUTPUT);

En esta línea, declaramos al pin digital 5 como salida.

analogWrite (5, LOW);

Enviamos al pin 5 un estado bajo o 0 voltios, led apagado.

brillo ();

Llamamos a la función brillo para que genere el efecto.

void brillo () {

for (int i=0; i<=255; i=i+10) {

analogWrite (5, i);

delay (200);

}

La función brillo() está compuesta por una serie de instrucciones que se detallan a continuación:

for (int i=0; i<=255; i=i+10) {

Creamos un bucle for, que incrementa de 10 en 10 a la variable i hasta llegar a un valor menor o igual a 255.

analogWrite (5, i);

El led empezará a encenderse, según el valor de la variable i.

delay (200);

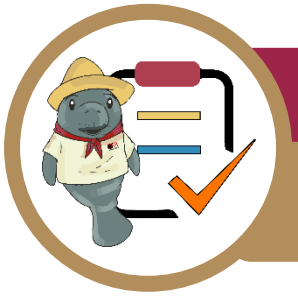
Espera 200 milisegundos y realiza la siguiente pasada por el bucle for.

Una modificación de esta práctica podría consistir en recrear el efecto completo, es decir, hacer que el led, una vez encendido completamente, vaya disminuyendo su luminosidad hasta apagarse, y así sucesivamente.

Copia este código al IDE de Arduino y súbelo a la placa, conecta todos los componentes de acuerdo al diagrama proporcionado anteriormente y contesta las siguientes preguntas:

- ¿Qué tipo de datos se utilizaron en el código del programa?
- ¿Identificas instrucciones que ya conocías? Menciónalas
- ¿Consideras importante el uso del PWM?

Realiza un reporte de la práctica debidamente estructurado y entrega al docente.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 9

Práctica 8: Luminosidad variable

DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Reporte de práctica		**Fecha**	
Formación laboral básica: Robótica **Submódulo 1:** Microcontroladores y microprocesadores		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Conecta correctamente el circuito con el LED y resistencia en el pin correspondiente.			2		
2.	Copia y carga el código correctamente al IDE de Arduino sin errores.			1		
3.	El LED muestra una variación progresiva de luminosidad (efecto PWM de encendido).			2		
4.	Identifica correctamente los tipos de datos utilizados (por ejemplo, int).			1		
5	Reconoce e interpreta el uso de analogWrite y del ciclo for en el control de luminosidad.			1		
6	Explica adecuadamente la importancia del PWM en el control de salida analógica.			1		
7	Propone o describe una mejora al efecto (como apagado progresivo después del encendido).			1		
8	Entrega un reporte completo, claro y bien estructurado de la práctica.			1		
				CALIFICACIÓN		

Lectura 15. Conectividad serial

Sabemos que existen dos maneras en las que se lleva a cabo la comunicación binaria, la paralela y la serial. En la comunicación paralela los datos viajan a través de varios hilos (cables), teniendo como ventaja una comunicación más rápida, pero también existen algunos inconvenientes como que, tantos hilos de transmisión pueden generar capacitancia por lo que la longitud de estos conductores solo puede ser de unos cuantos metros.

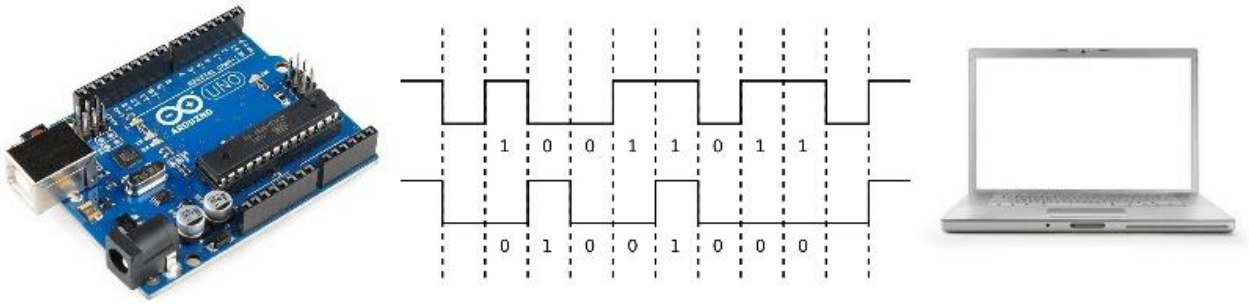
En la comunicación serial en cambio, se puede extender la comunicación a mayor distancia, con una menor cantidad de conductores, pero en contrapeso que la transmisión de datos es más lenta (ya que se transmite bit por bit) que en la comunicación en paralelo.

La longitud que pueden tener los conductores en la comunicación en serie depende de la norma que se esté utilizando para la transmisión, por ejemplo; en la norma RS232 a 15 mts., en la norma RS422/485 a 1200 mts. y utilizando un modem, a cualquier parte del planeta.

Existen dos formas en las que se puede llevar a cabo la comunicación serial, de forma síncrona y asíncrona. En la síncrona además del cable por donde se transmitirán lo datos, se necesita otro que contengan unos pulsos de reloj, y juntos determinan cuando un dato es válido. En la asíncrona no se necesita de pulsos de reloj, sino se utiliza un mecanismo de referencia a tierra (RS232) o voltajes diferenciales (RS422/485), "en donde la duración de cada bit es determinada por la velocidad de transmisión de datos que se debe definir previamente entre ambos equipos" (Reyes, 2008, p. 127).

Comunicación serial en Arduino

Todas las placas Arduino ofrecen la posibilidad de comunicar la placa con el ordenador a través del puerto serie o USB que incluyen. Gracias a dicha comunicación, tu placa de Arduino va a poder enviar y recibir información con el ordenador al que lo tienes conectado.



Arduino posee un conjunto de instrucciones relacionadas con la gestión de la comunicación de la placa con el ordenador a través del puerto serie.

Las instrucciones que Arduino tiene para trabajar con el puerto serie están dentro del tipo Serial. Mediante dicho tipo vas a poder realizar todas las funciones disponibles en Arduino para interactuar con el puerto serie.

El conjunto de instrucciones que ofrece Serial para realizar proyectos utilizando la comunicación serie con el ordenador pueden agruparse en dos grupos claramente diferenciados, aquellas instrucciones que permiten enviar información al puerto serie y aquellas instrucciones que permiten recibir información del puerto serie, además de las instrucciones para la inicialización y parada del canal de comunicaciones.

A continuación, se explican todas las instrucciones relacionadas con la comunicación serie que proporciona la clase Serial de Arduino.

begin()

La instrucción **begin** te permite inicializar el canal serie para que pueda comenzar la comunicación entre la placa y el ordenador. Es obligatorio realizar la inicialización antes de realizar cualquier tipo de comunicación mediante el puerto serie, en caso de no realizarse, la comunicación fallará.

begin utiliza el siguiente parámetro de entrada para su ejecución:

- *Velocidad de comunicación:* Entero que especifica la velocidad de comunicación entre la placa de Arduino y el ordenador. La velocidad normal de comunicación es 9600, aunque puedes modificarla, únicamente tienes que asegurarte que en la herramienta "Monitor Serie" del aplicativo de Arduino seleccionas la misma velocidad que tienes configurada en tu programa, ya que, en caso contrario, la velocidad estará desincronizada y aparecerán caracteres raros en la consola del monitor.

La instrucción **begin** no devuelve nada como resultado de su ejecución.

Sintaxis de la instrucción:

```
Serial.begin(9600);
```

end()

La instrucción end te va a permitir finalizar la comunicación serie entre la placa de Arduino y el ordenador.

end no utiliza ningún parámetro de entrada para poder ejecutarse y no devuelve nada.

Instrucciones de envio

El conjunto de instrucciones para el envío de datos a través del puerto serie sigue el siguiente formato:

Serial.Nombreinstruccion(InformacionAEnviar);

Las instrucciones son las siguiente:

- Serial.print: Instrucción que envía la información parametrizada al canal serie. El tipo de dato que se pasa como parámetro puede ser de cualquier tipo, pero puede añadirse un segundo parámetro opcionalmente que indica el formato de la información a enviar:

  • BIN: Información en formato binario.

  • OCT: Información en formato octal.

  • DEC: Información en formato decimal.

  • HEX: Información en formato hexadecimal.

  • Para números decimales, el segundo parámetro se puede utilizar para indicar e l número de decimales que se quieren enviar.

  Ejemplo:

```
int b = 79;
Serial.print(b, DEC);
```

- Serial.println: Instrucción parecida a la instrucción anterior, con el mismo comportamiento y finalidad, pero añade de forma automática un salto de línea al final de la información.
  Ejemplo:

```
Serial.println(analogRead(0)); // envía valor analógico
```

- Serial.write: Instrucción que envía información que ocupa exactamente un byte o una serie de bytes. La información para enviar puede ser de tipo byte, un array de bytes o un String.

Todas las instrucciones tienen en común las siguientes características:

- El envío de la información se realiza de forma asíncrona, es decir, el programa ejecutará el comando y continuará ejecutándose sin esperar a que empiece el envío. Este comportamiento puede cambiarse ejecutando la sentencia Serial.flush() justo después de la instrucción de envío.
- El valor de retorno indica el número de bytes enviados.

Instrucciones de recepción

El conjunto de instrucciones relacionadas con la recepción de información desde el ordenador a la placa son las siguientes:

Serial.available: Instrucción que devuelve el número de bytes disponibles para ser leídos por la placa de Arduino. Los bytes son almacenados temporalmente en un pequeño buffer de 64bytes.

Serial.read: Instrucción que realiza la lectura del primer byte del buffer y lo devuelve después de ejecutarse. El byte es eliminado del buffer. En caso de que no haya bytes que leer, la instrucción devolverá -l.

Ejemplo:

```
int incomingByte = 0; // almacena el dato serie
void setup() {
 Serial.begin(9600); // abre el puerto serie, y le asigna la velocidad de 9600 bps
}
void loop() {
// envía datos sólo si los recibe:
 if (Serial.available() > 0) {
// lee el byte de entrada:
 incomingByte = Serial.read();
//lo vuelca a pantalla
 Serial.print("He recibido: "); Serial.println(incomingByte, DEC);
 }
}
```

Serial.peek: Instrucción que realiza la lectura del primer byte del buffer y lo devuelve después de ejecutarse. El byte no es eliminado del buffer, por tanto, estará disponible en futuras lecturas que se hagan del buffer. En caso de que no haya bytes que leer, la instrucción devolverá -ll.Serial.find: Instrucción que realiza lecturas en el buffer hasta encontrar la cadena de texto indicada por parámetro. En caso de haber encontrado la cadena, la instrucción devuelve TRUE, en caso contrario devuelve FALSE. La instrucción no devuelve los datos leídos del buffer, únicamente los va eliminando hasta encontrar la cadena o se quede sin bytes que leer.

Serial.finduntil: Instrucción idéntica a la anterior, pero incorporando un parámetro más que indica una cadena de final de búsqueda. Es decir, esta instrucción parará cuando encuentre el texto a buscar, cuando llegue al final de la cadena o cuando encuentre la cadena de final de búsqueda.

Serial.readBytes: Instrucción que realiza la lectura del buffer de un número determinado de bytes, eliminándolos de allí, y devolviéndolos junto con el número de bytes leídos.

Serial.readBytesUntil: Instrucción idéntica a la anterior, pero incorporando un parámetro nuevo que especificará una cadena de texto que si es encontrada parará la lectura.

Serial.readString: Instrucción que realiza la lectura del buffer y los devuelve en formato String, eliminándolos del buffer. La instrucción deja de ejecutarse una vez ha expirado el tiempo establecido en la siguiente instrucción (Serial.setTimeout).

Serial.setTimeout: Instrucción que especifica mediante parámetro el número de milisegundos que las instrucciones Serial.readBytes, Serial. readBytesUntil y Serial.readString se quedan esperando la llegada de

nuevos datos al buffer de entrada. Una vez superado el tiempo, la instrucción que esté esperando es finalizada.

Serial.parseFloat: Instrucción que realiza la lectura del buffer hasta encontrar un número decimal, eliminando todos los bytes previos del buffer y devolviendo el número decimal encontrado.

Serial.parseInt: Instrucción igual que la anterior, pero con un número entero en lugar de un número decimal.

Actividad 4. Instrucciones de recepción serial

Instrucciones: Realiza una investigación bibliográfica de las aplicaciones de cada una de las siguientes instrucciones para la recepción de información de tipo serial, añadiendo ejemplos para reforzar la comprensión de su utilidad.

- *Serial.available*
- *Serial.read*
- *Serial.peek*
- *Serial.find*
- *Serial.finduntil*
- *Serial.readBytes*
- *Serial.readBytesUntil*
- *Serial.readString*
- *Serial.setTimeout*
- *Serial.parseFloat*
- *Serial.parseInt*

Actividad 5. Conversiones entre sistemas numéricos

Instrucciones: Después de observar y escuchar atentamente al docente con la presentación electrónica 4 "Sistemas numéricos y conversión entre sistemas", resuelve los siguientes ejercicios:

a) **Expresar el número decimal 67.924 como suma de los valores de cada dígito.**

b) **Convertir el número binario 10010001 a decimal.**

c) **Convertir el número binario 10,111 a decimal.**

d) **Convertir a binario el número decimal 39.**

e) **Convertir el número binario 1111011110011100 a hexadecimal.**

f) **Convertir el número hexadecimal 6BD3 a binario.**

g) **Convertir el numero hexadecimal 60A a decimal.**

h) Convertir a hexadecimal el número decimal 2591.

i) Convertir el numero octal 371 a decimal.

j) Convertir el numero decimal 2895 a octal.

k) Convertir el numero octal 140 a binario.

l) Convertir el numero binario 101111001 a octal.

Lectura 16. Compuertas lógicas y buffer

Compuertas lógicas

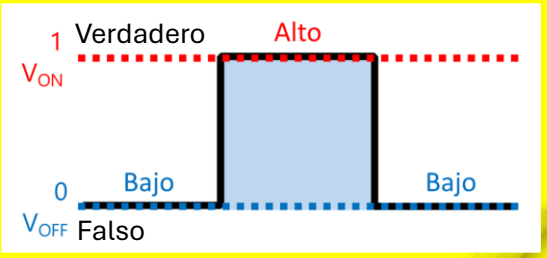
Las compuertas lógicas son elementos de electrónica digital que permiten realizar operaciones lógicas entre cantidades binarias. El álgebra de Boole es el área matemática específica que estudia las operaciones lógicas y sus propiedades. La implementación práctica de funciones lógicas se realiza mediante dispositivos electrónicos denominados compuertas lógicas, siendo éstas los componentes básicos de la electrónica digital.

Las compuertas lógicas proporcionan, generalmente en su salida, unos niveles de tensión en función de las tensiones presentes en sus entradas. Estos niveles son diferentes según la tecnología que se haya empleado en el proceso de fabricación, variando de unos dispositivos a otros. El conocimiento preciso de estos valores de tensión no es significativo en las operaciones lógicas, sino los rangos de tensiones entre los que operan las entradas y salidas de una compuerta lógica, llamados niveles lógicos. Así, existe un rango de tensiones alto (High), denominado nivel lógico alto y un rango de tensiones bajo (Low), denominado nivel lógico bajo.

Los resultados de un circuito lógico están determinados por un concepto denominado "Tabla de verdad", la cual es una tabla que ofrece un valor lógico de salida para una combinación de entrada.

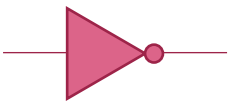
Dato interesante:

Cuando se habla de operaciones lógicas, se habla de niveles de verdad. Un cero lógico equivale a falso y un uno lógico equivale a verdadero.



a) Compuerta NOT

A la compuerta NOT también se le conoce como inversor o negador, y esta permite realizar la operación lógica NOT, lo que se traduce en que si a la entrada de la compuerta existe un cero lógico, como salida se tendrá un uno lógico, y si a la entrada recibe un uno lógico, como salida existirá un cero lógico.



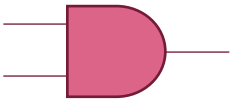
Entrada	Salida
0	1
1	0

Símbolo y tabla de verdad de la compuerta NOT.

En el algebra de Boole la compuerta NOT también tiene una expresión, en donde una entrada o variable, así como la salida se designan con una letra. Si la entrada de la compuerta NOT se denomina A y la salida X, la expresión Booleana de la operación lógica NOT es: $X = \bar{A}$

b) Compuerta AND

La compuerta AND permite realizar la operación lógica AND, donde como mínimo deben existir dos entradas y solo existirá una salida uno lógico si y solo si todas sus entradas son uno lógico.



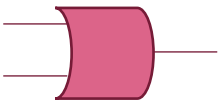
Entrada		Salida
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 7.2 Símbolo y tabla de verdad de la compuerta AND.

En el álgebra de Boole, la expresión de la compuerta AND es: $X = A \cdot B$

c) Compuerta OR

La compuerta OR permite realizar la operación lógica OR, donde como mínimo deben existir dos entradas y solo existirá una salida cero lógico si y solo si todas sus entradas son cero lógico.



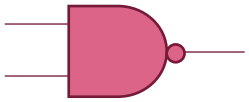
Entrada		Salida
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Fig. 7.3 Símbolo y tabla de verdad de la compuerta OR.

En el álgebra de Boole, la expresión de la compuerta OR es: $X = A+B$

d) Compuerta NAND

La compuerta NAND permite realizar la operación lógica NAND, donde como mínimo deben existir dos entradas y solo existirá una salida cero lógico si y solo si todas sus entradas son uno lógico.



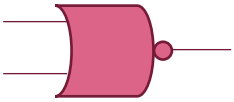
Entrada		Salida
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Fig. 7.4 Símbolo y tabla de verdad de la compuerta NAND.

En el álgebra de Boole, la expresión de la compuerta NAND es: $X = \overline{A \cdot B}$

e) Compuerta NOR

La compuerta NOR permite realizar la operación lógica NOR, donde como mínimo deben existir dos entradas y solo existirá una salida uno lógico si y solo si todas sus entradas son cero lógico.



Entrada		Salida
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Fig. 7.5 Símbolo y tabla de verdad de la compuerta NOR.

En el álgebra de Boole, la expresión de la compuerta NOR es: $X = \overline{A+B}$

Compuertas BUFFER

La compuerta BUFFER o seguidor solo tiene una entrada y ésta es igual a la salida, es decir, si la entrada vale 1, la salida también vale 1, y si la entrada vale 0 la salida también vale 0.

Como podemos darnos cuenta esta compuerta no ejecuta ninguna operación lógica sobre la entrada, pero "se justifica su utilización en aquellas aplicaciones en las que se requiere aumentar la corriente para excitar a dispositivos que así lo requieran" (Alegre y Caballero, 2002, p. 199).

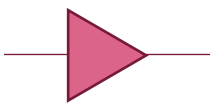


Fig. 7.6 Símbolo de la compuerta BUFFER.

Existe una compuerta BUFFER que posee una entrada de control adicional, por lo que también es llamado "Buffer de tres estados". Cuando la entrada de control cambia a estado alto, su salida cambia a un estado de alta impedancia, en este estado la salida se comportará como un interruptor abierto entre la salida del buffer y el bus o entrada.

Es necesario aclarar que, aunque la salida del buffer se encuentre en estado de alta impedancia, no tendrá ningún efecto sobre el nivel lógico del bus al cual este conectado. Cuando la entrada de control del Buffer se encuentre en un estado bajo, el buffer transferirá datos validos de su entrada a su salida.

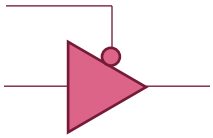
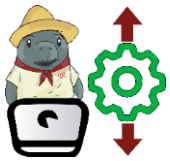


Fig. 7.7 Símbolo de la compuerta BUFFER de tres estados.

Si tomamos a la letra A como la entrada al buffer, H como la entrada de control, S como la señal de salida y Z como una señal de salida de alta impedancia, la tabla de verdad de un Buffer de tres estados es:

Entradas		Salida
A	**H**	**S**
0	0	0
0	1	Z
1	0	1
1	1	Z

Tabla. 7.6 Tabla de verdad de la compuerta Buffer de tres estados.



Práctica 9. Simulación de circuitos combinacionales

Instrucciones: Después de realizar la lectura "Compuertas lógicas y buffer" y escuchar atentamente la explicación del docente con los temas "Compuertas lógicas" y "Circuitos combinacionales", organízate en equipo de 5 integrantes para realizar la siguiente práctica, lee cuidadosamente cada uno de los puntos.

1. Realizar en el software de simulación de circuitos Proteus un circuito lógico combinacional de acuerdo con la siguiente expresión booleana. (Utiliza las compuertas adecuadas).

$$X=\overline{AC}\overline{(ABD)} + A\overline{BCD} + \overline{ABC}$$

2. Una vez realizado el circuito en Proteus, ir simulando con las combinaciones posibles en las entradas para ir construyendo la tabla de verdad correspondiente.

3. Simplificar la expresión del punto 1 aplicando el álgebra de Boole y los teoremas de DeMorgan.

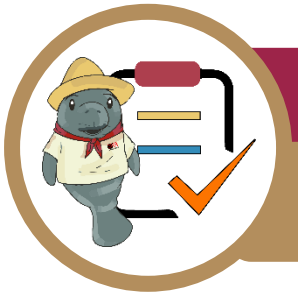
4. De acuerdo con la expresión simplificada, realizar en el simulador Proteus el circuito correspondiente.

5. Simular el circuito simplificado con las combinaciones posibles para comparar si las salidas resultantes son iguales a las del primer circuito. (Apoyarse con la tabla de verdad realizada en el punto 2).

6. Si las salidas no coinciden, revisar nuevamente la expresión simplificada para solucionar los errores cometidos.

7. Realizar un reporte de la práctica donde se plasme paso a paso la realización de esta, deben incluir las imágenes de los circuitos combinacionales desarrollados en Proteus, la tabla de verdad, así como el procedimiento que siguieron para simplificar la expresión dada.

8. Entregar el reporte y el archivo de simulación de Proteus en una carpeta digital.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 10

Práctica 9: Simulación de circuitos combinacionales

DATOS GENERALES		
Nombre(s) del alumno (s)		**Matriculas(s)**
Producto: Reporte de práctica		**Fecha**
Formación laboral básica: Robótica **Submódulo 1:** Microcontroladores y microprocesadores		**Periodo: 2026-2027A**
Nombre del docente:		**Firma del docente**

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Se realizó correctamente el primer circuito combinacional en Proteus según la expresión original			2		
2.	Se construyó la tabla de verdad completa con las combinaciones posibles de entrada			1		
3.	Se aplicó correctamente el álgebra de Boole y los teoremas de DeMorgan para simplificar la expresión			2		
4.	Se desarrolló en Proteus el circuito correspondiente a la expresión simplificada			1		
5	Se compararon correctamente los resultados de ambos circuitos mediante la tabla de verdad			1		
6	Se corrigieron errores en caso de discrepancias entre ambas salidas			1		
7	El reporte incluye imágenes de ambos circuitos, tabla de verdad y procedimiento de simplificación			2		
				CALIFICACIÓN		

Lectura 17. Diagnóstico de fallas de circuitos combinacionales

Tener conocimientos sobre el modo de operación de las entradas y salidas de los circuitos combinacionales, es útil cuando se quiere localizar una falla en estos sistemas, ya que se conocen los niveles lógicos a los que trabajan gracias a las tablas de verdad de los circuitos.

Los sistemas digitales (circuitos lógicos combinacionales) poseen en su estructura unos puntos de conexión entre sus entradas y salidas a los que se les denomina nodos.

Se llama *puerta excitadora* a la compuerta que se encarga de colocar a un nodo en un estado lógico bajo o alto, y *cargas* a las compuertas que están conectadas a un nodo a través de sus entradas.

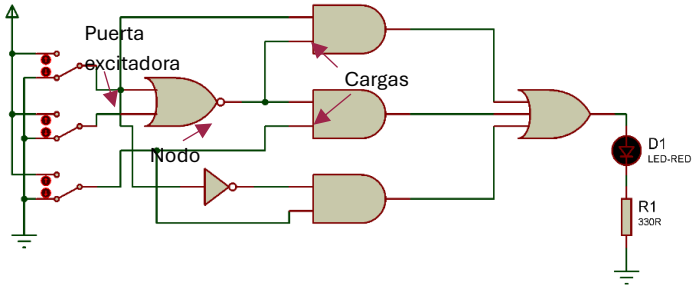


Fig. 8.1 Nodo en un circuito combinacional.

Debido a que dentro de un circuito pueden existir muchos nodos, esta situación puede producir diversos tipos de fallas, de las cuales las más comunes son:

1. *Salida en circuito abierto en la puerta excitadora.* Este fallo da lugar a pérdida de la señal en todas las puertas de carga.
2. *Entrada en circuito abierto en una puerta de carga.* Este fallo no afectará al funcionamiento de ninguna otra puerta conectada al nodo, pero hará que no se detecte señal de salida en la puerta que falla.
3. *Salida cortocircuitada de la puerta excitadora.* Este fallo puede dar lugar a que el nodo permanezca en estado BAJO (cortocircuitado a masa) o en estado ALTO (cortocircuitado a VCC).
4. *Entrada cortocircuitada en una puerta de carga.* Este fallo también hace que el nodo se mantenga a nivel BAJO (cortocircuitado a masa) o en estado ALTO (cortocircuitado a VCC).

Cuando se están localizando fallas en circuitos lógicos, es recomendable realizar primero una inspección visual con el objetivo de localizar los problemas obvios. En esta inspección den incluirse los conectores que llevan las señales, la alimentación y masa, ya que las superficies de contacto de estos conectores deben estar limpias, y en caso de que no, limpiarlas con un borrador de lápiz y un bastoncillo humedecido de alcohol.

Características para localizar las fallas más comunes.

Salida en circuito abierto en la puerta excitadora.

En este caso no se detectan impulsos en el nodo. Con el circuito alimentado, un nodo en circuito abierto dará lugar, normalmente, a un nivel "flotante", lo que puede indicarse mediante ruido.

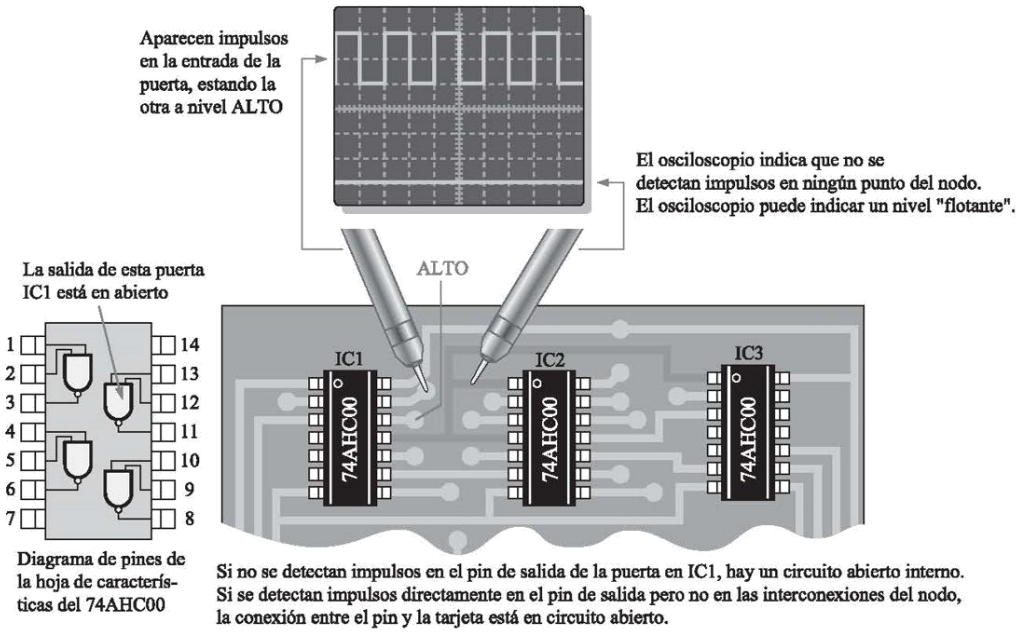


Fig. 8.2 Salida en circuito abierto en la puerta excitadora. Para simplificar,

se supone que hay un nivel ALTO en la entrada de una puerta (Floyd, 2006, p. 304).

Entrada en circuito abierto en una puerta de carga.

Si la salida de la puerta excitadora no está en circuito abierto, entonces hay que probar si la entrada de una puerta de carga está en circuito abierto. Permaneciendo las entradas de las puertas no pertenecientes al nodo a nivel ALTO, se comprueba con el osciloscopio la salida de cada una de las puertas. Si una de las entradas conectadas al nodo está en circuito abierto, no se detectarán impulsos en la salida de esta. Se puede apoyar de las tablas de verdad de las compuertas.

Salida o entrada cortocircuita a masa.

Si la salida está cortocircuitada a masa en la puerta excitadora o la entrada a una puerta de carga está cortocircuitada a masa, esto hará que el nodo permanezca a nivel BAJO, como se ha dicho anteriormente. Un cortocircuito a masa en la salida de la puerta excitadora o en la entrada de cualquier puerta de carga

dará lugar a este síntoma, por lo que deben hacerse más comprobaciones para aislar el cortocircuito en una puerta en concreto.

Análisis y seguimiento de señales

Aparte del método de aislamiento de cortocircuitos y circuitos abiertos en un nodo, existe otro método más general llamado seguimiento de señales, donde la medida de la señal se realiza con un osciloscopio o un analizador lógico. "Básicamente, el método de seguimiento de señales requiere que se observen las formas de onda y sus relaciones temporales en todos los puntos accesibles del circuito lógico" (Floyd, 2006, p. 305).

Según Floyd (2006), el procedimiento general del seguimiento de señales comenzando por las entradas es el siguiente:

- Dentro del sistema, definir la sección del circuito lógico que se sospecha que está fallando.
- Comenzar en las entradas de la sección que se va a examinar. Para este estudio, suponemos que las formas de onda de entrada proceden de otras partes del sistema que son correctas.
- Para cada puerta, empezando por la entrada y yendo hacia la salida del circuito lógico, se observa la forma de onda de salida de la puerta y se compara con las formas de onda de entrada, utilizando el osciloscopio o el analizador lógico.
- Determinar si la señal de salida es correcta utilizando nuestros conocimientos sobre la operación lógica de la puerta.
- Si la salida es incorrecta, en la puerta bajo prueba puede estar el fallo. Extraiga el CI que contiene la puerta de la que se sospecha que produce el fallo, y compruébelo fuera del circuito. Si la puerta falla, reemplace el CI. Si funciona correctamente, el fallo está en la circuitería externa o en otro CI al que está conectado el que se está probando.
- Si la salida es correcta, pase a la puerta siguiente. Continúe comprobando cada puerta hasta observar una forma de onda incorrecta.

Recurso didáctico sugerido

¿Cómo usar el osciloscopio?

https://www.youtube.com/watch?v=2U-mR62OVUg

Actividad 6. Seguimiento de señales

Instrucciones: Después de realizar la lectura "Diagnóstico de fallas de sistemas digitales", escuchar la explicación del docente y apoyado del "Ejemplo de seguimiento de señales para diagnóstico de fallas" proporcionado por el docente, resuelve la situación que se presenta a continuación.

1. La forma de onda de salida de la Figura 8.3 es incorrecta para las entradas que se aplican al circuito. Suponiendo que una puerta del circuito está fallando, con su salida a un nivel ALTO o BAJO constante, determinar la puerta que falla y el tipo de fallo (circuito abierto o cortocircuito).

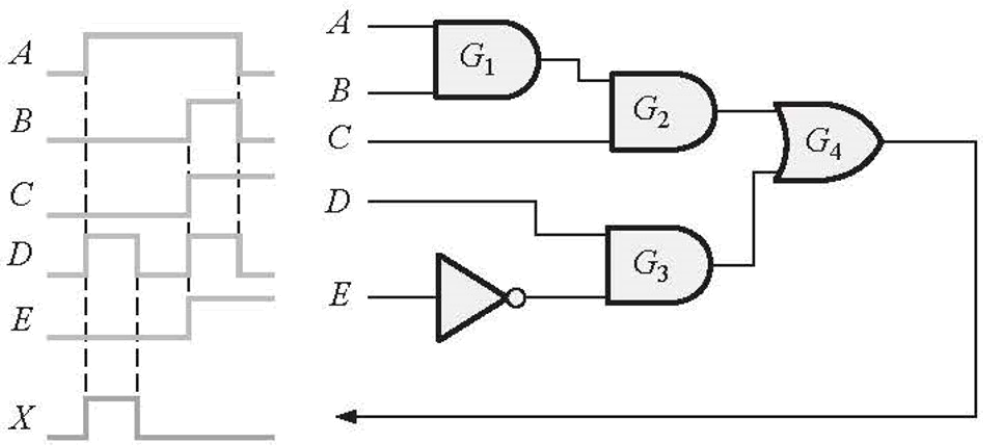
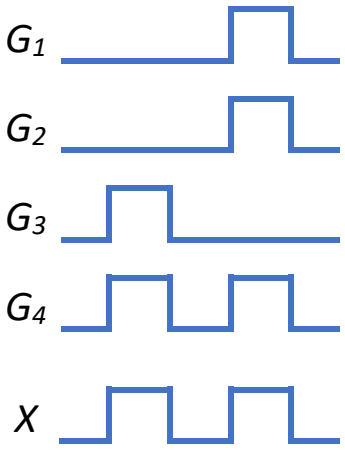


Fig. 8.3 Circuito para análisis de señales.

a) Señales de salida de las compuertas G1, G2, G3 Y G4



b) Análisis para determinar la puerta que falla, suponiendo que está fallando con su salida a un nivel ALTO o BAJO constante.

Lectura 18. Funciones de la lógica combinacional

Las funciones de los circuitos lógicos combinacionales pueden ser muy variada, dependiendo de las aplicaciones para los que serán destinados, así podemos encontrar sumadores, comparadores, decodificadores, codificadores, convertidores de código, multiplexores (selectores de datos), demultiplexores, generadores/comprobadores de paridad, circuitos integrados de función fija, etc.

Sumadores básicos.

En los sistemas digitales que procesan datos numéricos la aplicación de los sumadores es muy relevante, es por ello por lo que se debe comprender el funcionamiento de sumadores básicos.

El semi-sumador.

Para construir un semisumador básico es necesario conocer las reglas de la suma binaria, en donde se tienen cuatro posibilidades: 0+0= 0; 0+1=1; 1+0=1; 1+1=10

Un semi-sumador suma dos bits y genera una suma y una salida de acarreo. La salida de acarreo se produce mediante una puerta AND, y la salida de la suma se obtiene mediante una puerta OR-exclusiva.

A	B	C	∑
0	0	0	1
0	1	0	0
1	0	0	0
1	1	1	1

∑= suma
C= acarreo
A y B= variables de entrada

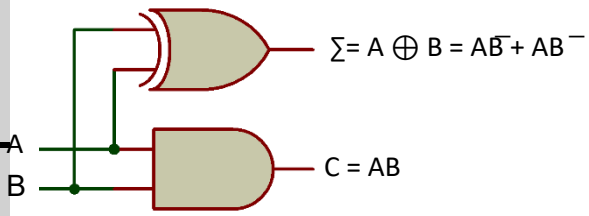


Fig. 9.1 Tabla de verdad y diagrama lógico de un semi-sumador.

El sumador completo.

El sumador básico completo además de tener todos los elementos del semisumador, también posee un acarreo de entrada.

Del semisumador sabemos que la suma de A y B se obtiene al aplicar la operación OR-exclusiva de esas dos variables, pero en el sumador completo, para sumar el acarreo de entrada, se tiene que aplicar de nuevo la operación Or-exclusiva.

Esto significa que para implementar la función del sumador completo se pueden utilizar dos puertas OR-exclusiva de 2 entradas. La primera tiene que generar el término A ⊕ B, y la segunda tiene como entradas la salida de la primera puerta XOR y el acarreo de entrada.

A	B	CI	CO	∑
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

∑= suma
CI= acarreo de entrada
CO= acarreo de salida
A y B= variables de entrada

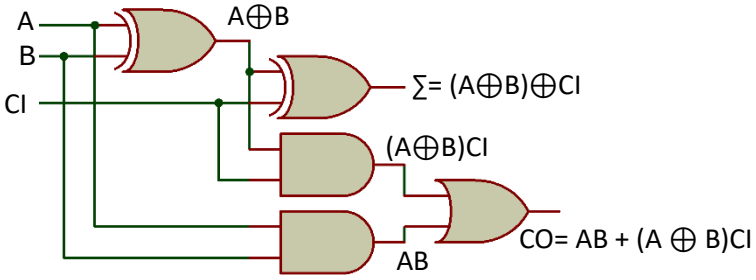


Fig. 9.2 Tabla de verdad y diagrama lógico de un sumador completo.

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 OR-exclusiva 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.

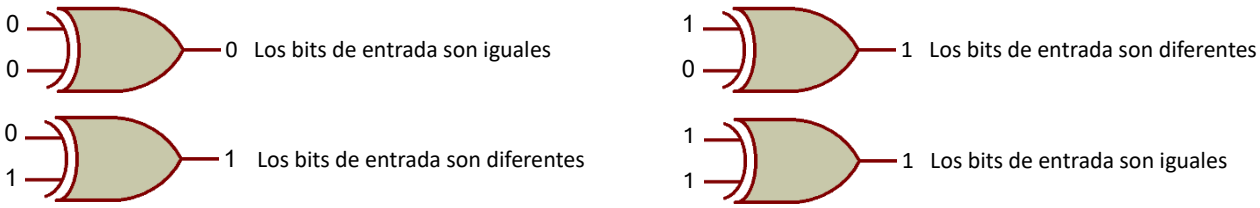


Fig. 9.3 Funcionamiento del comparador básico de 2 bits.

Decodificadores

La función básica de un decodificador es detectar la presencia de una determinada combinación de bits (código) en sus entradas y señalar la presencia de este código mediante un cierto nivel de salida. En su

forma general, un decodificador posee n líneas de entrada para gestionar n bits y en una de las 2^n líneas de salida indica la presencia de una o más combinaciones de n bits.

El decodificador binario básico

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. 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).

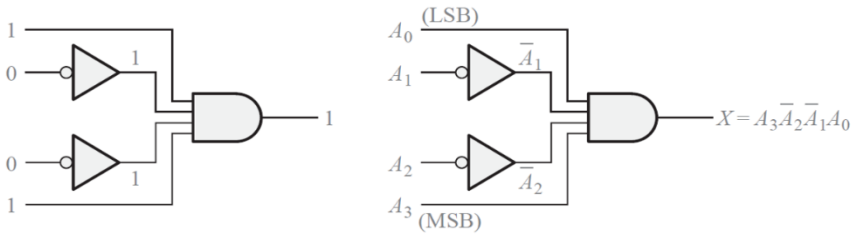


Fig. 9.4 Lógica de decodificación del código binario 1001 con una salida activa a nivel ALTO (Floyd, 2006).

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).

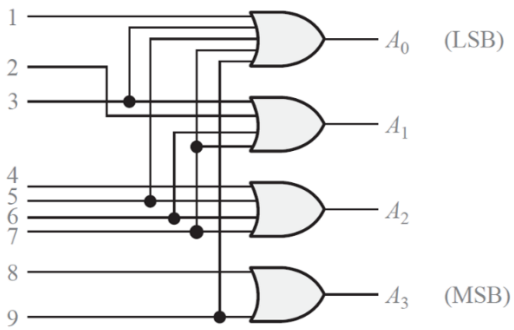


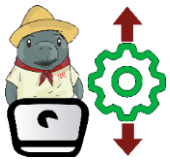
Fig. 9.5 Diagrama lógico básico de un codificador decimal-BCD. No se necesita una entrada para el dígito 0, ya que las salidas BCD están todas a nivel BAJO cuando no hay entradas a nivel ALTO. (Floyd, 2006).

Multiplexores (selectores de datos)

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 a un destino común. 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.

Demultiplexores

Un demultiplexor (DEMUX) básicamente realiza la función contraria a la del multiplexor. Toma datos de una línea y los distribuye a un determinado número de líneas de salida. Por este motivo, el demultiplexor se conoce también como distribuidor de datos.



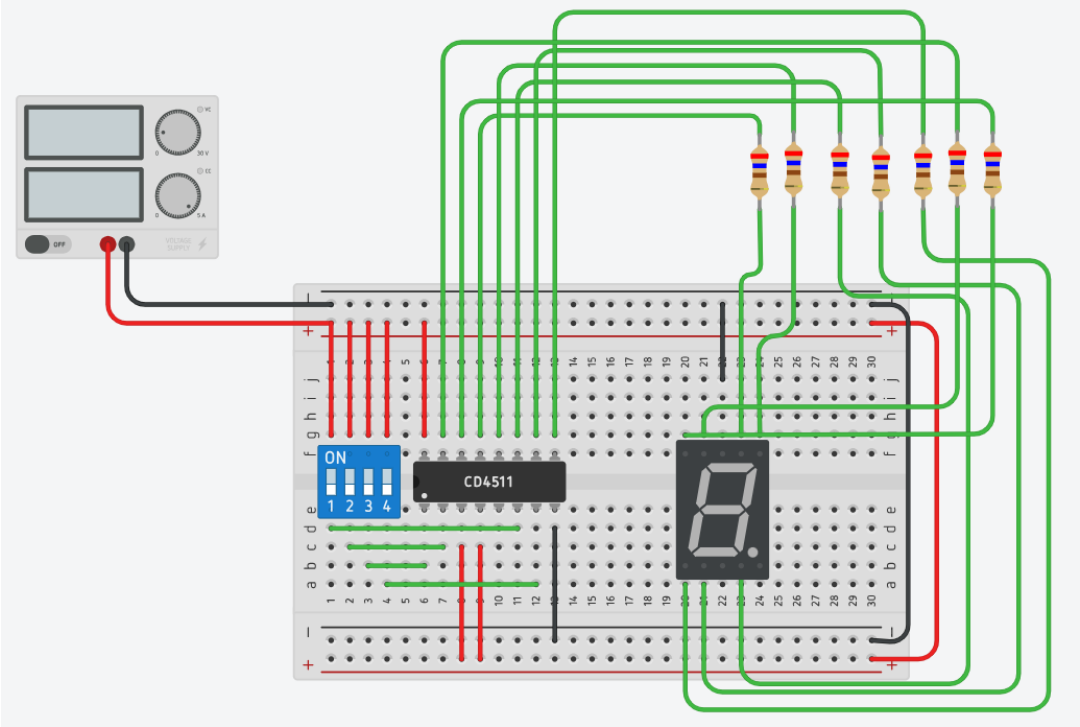
Práctica 10. Decodificador

Instrucciones: Lee con atención lo que se indica en la siguiente práctica, cumpliendo con cada uno de los puntos que se especifican.

Objetivos:

- El decodificador CD4511 convierte un número en código BCD (Binary Coded Decimal) en señales adecuadas para encender un display de 7 segmentos. Esta práctica consiste en montar un circuito que muestre números del 0 al 9 de forma visual, utilizando interruptores.
- Comprender el funcionamiento de un decodificador BCD a 7 segmentos.
- Conectar correctamente el circuito integrado CD4511 para visualizar números en un display.
- Relacionar los conceptos de codificación, decodificación y visualización digital.
- Reforzar habilidades en el armado de circuitos digitales con componentes integrados.

Esquema de conexión:



Lista de materiales:

1 circuito integrado CD4511
1 display de 7 segmentos cátodo común
4 interruptores (Dip switch de 4 entradas) para representar entradas BCD (A, B, C, D)
7 resistencias de 330 Ω para el display.
Protoboard
Cables para conexiones
Fuente de alimentación de 5V

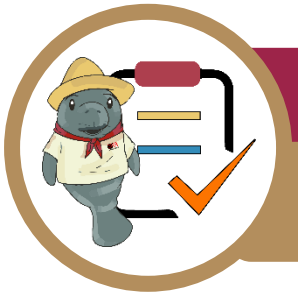
Procedimiento.

- Identifica los pines del CD4511 y del display de 7 segmentos (busca su hoja de datos en la web).
- Conecta los pines A, B, C y D del CD4511 a cuatro interruptores que simulen las entradas BCD.
- Conecta las salidas a-g del CD4511 a los pines correspondientes del display.
- Conecta resistencias limitadoras de 330 Ω entre las salidas del CD4511 y el display.
- Asegúrate de conectar el polo común del display al GND, ya que es un cátodo común.
- Conecta el pin LE (Latch Enable) del CD4511 a tierra y los pines LT (Lamp Test) y BI (Blank Input) a Vcc (para habilitar la salida).
- Verifica que, al accionar los interruptores, se visualicen los dígitos del 0 al 9 en el display.

Preguntas para reflexión y reforzamiento.

1. ¿Qué función cumple el CD4511 en este circuito?
2. ¿Por qué es importante que el display sea de cátodo común para este circuito?
3. ¿Qué sucede si se introduce una combinación binaria mayor a 1001 (9 en BCD)? ¿Se muestra un dígito válido?
4. ¿Cuál es la diferencia entre codificación y decodificación?
5. ¿Cómo podrían usarse múltiples decodificadores como el CD4511 en una aplicación real?
6. ¿Por qué se utilizan resistencias en serie con los LEDs del display?
7. ¿Qué ventajas tiene usar un decodificador en vez de controlar manualmente cada segmento?

Realiza un reporte de la práctica debidamente estructurado y entrega al docente.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 11

Práctica 10: Decodificador

DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Reporte de práctica		**Fecha**	
Formación laboral básica: Robótica **Submódulo 1:** Microcontroladores y microprocesadores		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Conecta correctamente el CD4511 según el diagrama (alimentación, entradas y salidas).			2		
2.	Conecta el display de 7 segmentos de cátodo común correctamente con resistencias limitadoras.			2		
3.	El circuito muestra correctamente los números del 0 al 9 según la entrada BCD.			2		
4.	Identifica y explica el propósito de los pines LT, BI y LE del CD4511.			1		
5	Responde correctamente las preguntas de reflexión con base en observaciones y teoría.			2		
6	Entrega un reporte claro, ordenado y completo, incluyendo fotos o diagrama del circuito			1		
				CALIFICACIÓN		

Situación didáctica 1
Midiendo temperatura y humedad

Instrucciones: Con base en la situación didáctica planteada al principio del módulo, plantea el prototipo de un robot contenedor donde aplique todos los conocimientos aprendidos durante el desarrollo del submódulo.

Propósito de la Situación didáctica 1. En equipo de 6 integrantes utilizando la aplicación de Tinkercad y de ser posible de forma física con los materiales necesarios, realizar el Diseño, Programación, Simulación y/o puesta en marcha física de un Sensor de Humedad y Temperatura para poder ser aplicado en un contexto de huerto sostenible en la comunidad educativa.

Conocimientos:

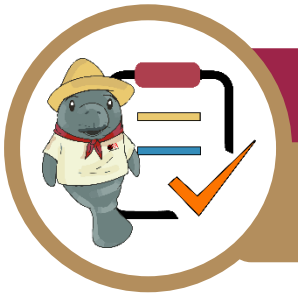
1. Microcontroladores y microprocesadores
2. Arquitectura de un microcontrolador
3. Compiladores para microcontroladores
   - 3.1 Estructura básica de un programa
   - 3.2 Tipos de datos y variables
   - 3.3 Estructuras de control
4. Configuración de puertos de entrada y salida
5. Modo de operación Stand Alone
6. Contadores
7. Interrupciones
8. Temporizadores
9. PWM
10. Conectividad serial
11. Circuitos lógicos
    - 11.1 Sistemas numéricos y conversión entre sistemas
    - 11.2 Compuertas lógicas: AND, OR, NOT, NOR, NAND
    - 11.3 Compuertas BUFFER
12. Circuitos combinacionales
13. Diagnóstico de falla de circuitos
14. Funciones de lógica combinacional

Materiales

- Computadora
- Software de simulación de circuitos.
- Arduino uno
- Sensor de proximidad
- Motor dc
- Baterías 9v
- Software para escribir programas en Arduino
- Libreta de apuntes.
- Guía del estudiante.

Procedimiento:

En equipos de 6 integrantes organizar y plasmar el diseño y circuitos del robot, en un reporte todos los acuerdos tomados durante el transcurso del submódulo para presentar ante el grupo el planteamiento y diseño de un robot que cumpla con las características planteadas al inicio del submódulo, mismas que se encuentran plasmadas en la lista de cotejo.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 12

Situación didáctica 1: Midiendo temperatura y humedad

DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Prototipo		**Fecha**	
Formación laboral básica: Robótica **Submódulo 1:** Microcontroladores y microprocesadores		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	El equipo diseñó un prototipo funcional o simulado de robot con sensor de humedad y temperatura.			2		
2.	El prototipo está correctamente programado y cumple su función básica (medición y/o activación).			2		
3.	El prototipo está correctamente programado y cumple su función básica (medición y/o activación).			2		
4.	El equipo presentó un reporte claro con diseño del circuito, acuerdos y funcionamiento.			2		
5	Se integró la propuesta al contexto del huerto escolar con enfoque social y ambiental.			1		
6	Participación activa y colaborativa de todos los integrantes del equipo.			1		
				CALIFICACIÓN		

SUBMÓDULO II

CONTROL INDUSTRIAL

Propósito del Submódulo

Establece circuitos de acoplamiento de alta potencia para PLC, mediante la programación del mismo, para resolver de forma creativa diversas problemáticas del ámbito industrial y prevenir riesgos en situaciones cotidianas.

Competencias laborales básicas a desarrollar

- Selecciona en compañía de su docente programas para PLC mediante la codificación y compilación de las instrucciones pertinentes para cumplir con los requerimientos de funcionalidad y rendimiento establecidos de control industrial, actuando de forma congruente y consciente previniendo riesgos en situaciones cotidianas.

Evidencias de evaluación

- Propone programas en forma creativa y consciente, para PLC, apoyándose en dispositivos físicos o simuladores, en beneficio de su contexto.
- Desarrolla circuitos de acoplamiento de alta potencia para PLC, actuando de forma congruente y consciente para prevenir riesgos en situaciones cotidianas.
- Prepara piezas usando técnicas de modelado, fomentando el trabajo colaborativo y creativo, proponiendo soluciones para el sector industrial.
- Construye piezas a través de métodos de modelado para su fabricación e impresión en 3D o CNC, haciendo una evaluación crítica de su trabajo y resolviendo problemas cotidianos de forma creativa.

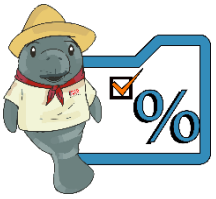
DOSIFICACIÓN PROGRAMÁTICA

Formación Laboral Básica: Robótica
Módulo III: Robótica industrial
Submódulo II: Control industrial **Clave:** C5CI
Semestre: 5to. **Turno:** Matutino/Vespertino **Periodo:** 2026 – 2027A

Submódulo	Momento	Tiempo (minutos)	Conocimientos	Semana	Fecha inicio	Observaciones
SUBMÓDULO II. CONTROL INDUSTRIAL	Apertura	350 min	Bienvenida Encuadre Reglamento Instrumentos de evaluación. Evaluación diagnóstica. **1. Controladores lógicos programables (PLC)** **2. Estructura interna** **3. clasificación** 4. **Puertos de entrada y salida**	**8**	19 – 23 de octubre de 2026	
	Desarrollo	350 n	**Puertos de entrada y salida** 5. **Manejo de instalación y conexiones**	**9**	26 – 30 de octubre de 2026	Reunión de academia
		350 min	**6. Programación básica en lenguaje escalera** 7. **Ejecución**	**10**	02 – 06 de noviembre de 2026	
		350 min	**8. Contadores** 9. **Temporizadores**	**11**	09 – 13 de noviembre de 2026	

Submódulo	Momento	Tiempo (minutos)	Conocimientos	Semana	Fecha inicio	Observaciones
SUBMÓDULO II. CONTROL INDUSTRIAL		250 min	**10. Simuladores de aplicaciones industriales**			
		100 min	**11. Circuitos de acoplamiento para alta potencia**	**12**	16 – 20 de noviembre de 2026	
	Desarrollo	100 min 250 min	**12. Dibujo industrial** **13. Diseño asistido por computadora** 13.1. Vistas 13.2. Croquis 13.3. Líneas	**13**	23 – 27 noviembre de 2026	
		350 n	13.4. Rectángulos 13.5. Círculos 13.6. Acotaciones	**14**	30 de noviembre – 04 de diciembre de 2026	
		350 n	13.7. Extrusión 3D y cortes , impresiones y uso del software	**15**	07 – 11 de diciembre de 2026	Evaluación sumativa
	Cierre	350 min	**14. Control numérico computarizado (CNC)** **Situación didáctica:** ¿Mi escuela, con robots industriales?	**16**	14 – 18 de diciembre de 2026	 Reunión de academia Reforzamiento académico
				17 y 18	21 de diciembre al 08 de enero 2026	Vacaciones Suspensión oficial 25 y 01 Retroalimentación
				19	12 al 15 de enero 2026	Evaluación Extraordinaria Intersemestral

				20	18 al 22 de enero 2026	Jornada de capacitación
				21 y 22	25 al 29 de enero 2026	Trabajos colegiados Fin de periodo
	Total:	5,600 min				



Encuadre Submódulo 2 "Control industrial"

Situación Didáctica	
Actividades	**Puntaje**
Actividad 1. Características de los PLC.	2%
Actividad 2. Puertos de entrada y salida PLC.	2%
Actividad 3. Conexiones, instalación y manejo de PLC.	2%
Actividad 4. Lenguaje escalera, popular en la programación de PLC.	2%
Actividad 5. Programando mi PLC en lenguaje escalera.	2%
Actividad 6. Contadores y temporizadores en Mi PLC.	2%
Actividad 7. Explorando el uso de simuladores industriales.	2%
Actividad 8. Dibujo industrial.	2%
Actividad 9. Explorando los estilos y presentaciones en Autocad.	2%
Actividad 10. Extrusión, cortes e impresiones en Autocad.	1%
Actividad 11. Control numérico computarizado (CNC).	1%
Actividad 1. Características de los PLC.	2%
Suma de actividades:	20%
Ejercicios	**Puntaje**
Ejercicio 1. Proyecto PLC en Tinkercad.	2%
Ejercicio 2. Simulación de un circuito lógico en Tinkercad.	2%
Ejercicio 3. Circuito de escalera en Tinkercad.	2%
Ejercicio 4. PLC en lenguaje escalera.	2%
Ejercicio 5. Temporizadores en lenguaje escalera.	2%
Ejercicio 6. Contadores en lenguaje escalera.	2%
Ejercicio 7. Modelo de potencia en Tinkercad.	2%
Ejercicio 8. Vistas y croquis en papel.	2%

Ejercicio 9. Vistas y croquis en Autocad.	2%
Ejercicio 10. Entidades de dibujo en autocad.	2%
Total	**20%**
Prácticas	**Puntaje**
Práctica 1. Control de velocidad básico de un motor DC con PWM y Arduino.	10%
Práctica 2. Simulando procesos industriales con Flexisim.	10%
Práctica 3. PLC en Autocad.	10%
Total	**30%**
Situación de Aprendizaje "¿Mi escuela, con robots industriales?"	**30%**
Total	100%

Situación Didáctica Submódulo 2

Situación Didáctica 2	
Título:	¿Mi escuela, con robots industriales?
Contexto:	Actualmente la robótica está muy especializada en el sector industrial, por ser indispensable en la mayoría de los procesos de manufactura, con ello las aplicaciones de la robótica industrial debido a su misma naturaleza es multifuncional, por llevar a cabo muchas tareas y es necesario estar dispuestos a admitir cambios en el desarrollo de los procesos, así como en la modificación de piezas y sustitución de sistemas. Acorde a esto el tipo de robot a utilizar debe considerar la velocidad de carga, la capacidad de control y aspectos relacionados con su propia funcionalidad. Por esto la finalidad es que los estudiantes tengan contacto con la robótica industrial en el diseño y desarrollo de piezas industriales, así como la programación y control de estas por medio de PLC
Conflicto Cognitivo	¿Qué son los controladores lógicos programables (PLC)? ¿Qué es la programación básica en lenguaje escalera? ¿Qué son los contadores y temporizadores? ¿Cuáles son los simuladores de aplicaciones industriales? ¿Qué es un circuito de acoplamiento para alta potencia? ¿Qué es un control numérico computarizado (CNC)? ¿Sabes para que sirve un robot industrial? ¿Cómo se programa un robot industrial? ¿Cuáles son las fases para construir un robot industrial?
Estrategia Didáctica	Robots industriales

Evaluación diagnóstica Submódulo 2

Instrucciones: Lee cuidadosamente los siguientes cuestionamientos y selecciona en base a tus conocimientos la respuesta que consideres correcta.

1. Indica. Básicamente es una especie de computadora que se utiliza para realizar tareas automatizadas, como pueden ser líneas de ensamblaje en fábricas, sistemas de iluminación o cualquier otro tipo de proceso que sea automatizable
   A) Control lógico programable
   B) Puerto de entrada y salida
   C) Estructura interna
   D) Conexiones

2. Elige. Año desde que existen los PLC.
   A) 1970
   B) 1980
   C) 1960
   D) 1990

3. Selecciona. Es en donde se encuentran alojados los distintos bloques que hacen posible el correcto funcionamiento del PLC.
   A) Software
   B) Bus
   C) Hardware
   D) Memoria

4. Indica. Compuesta de dispositivos electrónicos para poder alojar las instrucciones básicas del funcionamiento del PLC, así como las unidades para procesar instrucciones de un programa precargado y realizar las tareas especificadas en él.
   A) Hardware
   B) Externa
   C) Software
   D) Interna

5. Elige. Es lo correspondiente a los módulos de entradas y salidas digitales, fuente de poder, carcasa, indicadores led; contiene los elementos netamente tangibles del PLC.
   A) Hardware
   B) Externa
   C) Software
   D) Interna

6. Selecciona. Actúa como un contador, con la diferencia que no realiza el conteo de eventos externos, lo hace a través de un generador de pulsos o de frecuencia dentro de la CPU.
   A) Buses
   B) Conversor
   C) Temporizador
   D) Memoria

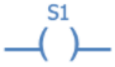
7. Elige. Destinados a leer datos analógicos y convertirlos a datos binarios.
   A) Buses
   B) Conversor
   C) Temporizadores
   D) Memoria

8. Elige. Son aquellos PLC que están conformados por CPUS, PS, y módulos de entrada y salida en un mismo compartimiento, disponen por igual, de una entrada donde se puede medir la alta velocidad, y a la par cuenta con dos controladores analógicos.
   A) Compactos
   B) Montaje en rack
   C) Modular
   D) Con panel operador

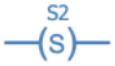
9. Selecciona. Cuenta con un interfaz que facilita y optimiza su funcionamiento, que brindan una supervisión constante y actividad de monitoreo a las actividades que se presentan en las maquinas.
   A) Compactus
   B) Montage en rack
   C) Modular
   D) Con panel operador

10. Elige. Este lenguaje de programación se programa mediante símbolos gráficos y en diferentes segmentos. En cada segmento, programamos las diferentes sentencias de la lógica.
    A) Lenguaje escalera
    B) Lenguaje Visual Basic
    B) Lenguaje Scratch
    C) Lenguaje C

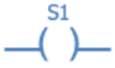
11. Indica. Es la entrada contacto normalmente abierto.
    A) 

    B) 

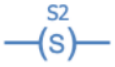
    C) 

    D) 

12. Elige. Es la salida, bobina o relé.
    A) 

    B) 

    C) 

    D) 

13. Selecciona. Son posiciones de memoria que almacenan un valor numérico, mismo que se incrementa o decrementa según la configuración dada.
    A) Contadores
    B) Relés
    C) Temporizadores
    D) Registros de desplazamiento

14. Indica. Se pueden usar, por ejemplo, para memorizar estados o como acumuladores de resultados que utilizaran posteriormente en el programa.
    A) Contadores
    B) Relés
    C) Temporizadores
    D) Registros de desplazamiento

15. Elige. Es una herramienta que permite reproducir virtualmente los procesos y estudiar su comportamiento, para analizar el impacto de las distintas variables que puedan intervenir en el mismo, o para comparar diferentes alternativas de diseño, sin el alto coste de los experimentos.
    A) Circuitos de acoplamiento C) Simuladores industriales
    B) CAD D) PLC

16. Indica. Está dada por el producto de la tensión y la corriente que caracterizan al dipolo en cuestión.
    A) Potencia instantánea C) Potencia reactiva
    B) Potencia activa D) Potencia aparente

17. Selecciona. Permite plasmar ideas y comunicar proyectos a través del uso de escalas, perspectivas y diversas **técnicas** de representación. En estos dibujos suele apelarse a símbolos para facilitar la inclusión de datos que sean fáciles de comprender para todos los profesionales.
    A) Dibujo artístico C) Dibujo geométrico
    B) Dibujo arquitectónico D) Dibujo industrial

18. Indica. Es un sistema que permite controlar en todo momento la posición de un elemento físico.
    A) Control numérico C) Control numérico computarizado
    B) Control industrial D) Control gráfico

Lectura 1. ¿Qué es un control lógico programable?

El PLC (Control Lógico Programable) apareció con el propósito de eliminar el enorme costo que significaba el reemplazo de un sistema de control basado en relés (relays) a finales de los años 60.

El MODICON 084 fue el primer PLC producido comercialmente. Este nuevo controlador (el PLC) tenía que ser fácilmente programable, su vida útil tenía que ser larga y ser resistente a ambientes difíciles. Esto se logró con técnicas de programación conocidas y reemplazando los relés por elementos de estado sólido.

Con el avance en el desarrollo de los microprocesadores (más veloces), cada vez PLC más grandes se basan en ellos. La habilidad de comunicación entre ellos apareció aproximadamente en el año 1973. El primer sistema que lo hacía fue el Modbus de Modicon. Los PLC podían incluso estar alejados de la maquinaria que controlaban, pero la falta de estandarización debido al constante cambio en la tecnología hizo que esta comunicación se tornara difícil.

En los años 80 se intentó estandarizar la comunicación entre PLCs con el protocolo de automatización de manufactura de la General Motors (MAP). En esos tiempos el tamaño del PLC se redujo, su programación se realizaba mediante computadoras personales (PC) en vez de terminales dedicadas sólo a ese propósito. En los años 90 se introdujeron nuevos protocolos y se mejoraron algunos anteriores. Ahora se tiene PLCs que se programan en función de diagrama de bloques, listas de instrucciones, lenguaje C, etc. al mismo tiempo.

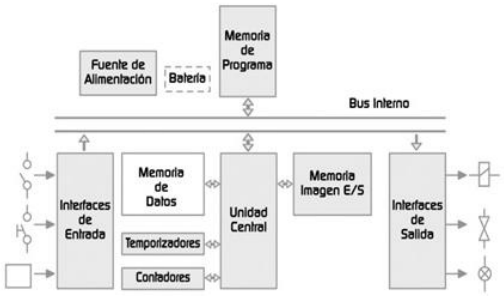
El PLC por sus especiales características de diseño tiene un campo de aplicación muy extenso. La constante evolución del hardware y software amplía constantemente este campo para poder satisfacer las necesidades que se detectan en el espectro de sus posibilidades reales. Su utilización se da fundamentalmente en aquellas instalaciones en donde es necesario un proceso de maniobra, control, señalización, etc., por tanto, su aplicación abarca desde procesos de fabricación industriales de cualquier tipo a transformaciones industriales, control de instalaciones, etc.

Sus reducidas dimensiones, la extremada facilidad de su montaje, la posibilidad de almacenar los programas para su posterior y rápida utilización, la modificación o alteración de estos, etc., hace que su eficacia se aprecie fundamentalmente en procesos en que se producen necesidades tales como: espacio reducido, procesos de producción periódicamente cambiantes y procesos secuenciales.

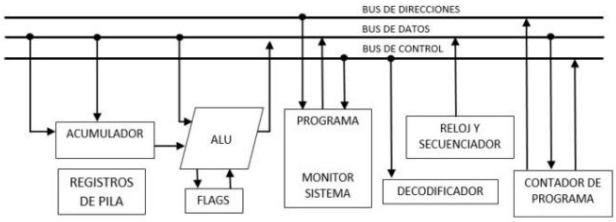
Ejemplos de aplicaciones industriales: maniobra de máquinas, maquinaria industrial de plástico, máquinas transfer, maquinaria de embalajes, maniobra de instalaciones: instalación de aire acondicionado, calefacción, instalaciones de seguridad. Señalización y control: chequeo de programas, señalización del estado de procesos.

Estructura interna de un PLC.

Compuesta de dispositivos electrónicos para poder alojar las instrucciones básicas del funcionamiento del PLC, así como las unidades para procesar instrucciones de un programa precargado y realizar las tareas especificadas en él. Como podemos observar, se muestra el diagrama de bloques correspondiente a la estructura interna del PLC. Podemos observar que se cuentan con arreglos de memorias destinados a alojar datos, programas, se cuenta con un procesador o unidad de control, interfaces de entrada y salida, buses de comunicación, temporizadores y contadores.



Unidad de Control (CPU): Destinada a consultar el estado de las entradas, analizar el programa cargado previamente y así poder escribir las instrucciones para la salida. El ciclo de scan del programa (lectura de entradas, lectura de programa y escritura de salidas) se realiza por default en 150 milisegundos, donde, el PLC traduce el programa a lenguaje máquina, realizando operaciones lógicas para realizar el proceso requerido.



Como se observa la CPU contiene:

- **ALU:** Realiza operaciones aritmeticológicas
- **Acumulador:** Almacena el último resultado de la ALU
- **Flags:** Indicadores de resultado (positivo, negativo, mayor, menor que)
- **Contador de Programa:** Lectura de instrucciones de usuario
- **Decodificador de Instrucciones y Secuenciador:** Lugar donde se decodifican las instrucciones y se generan las señales de control
- **Pila:** Prioriza las instrucciones a realizar, evitando saltos en el programa o en las instrucciones
- **Monitor Sistema:** Almacena la secuencia de puesta en marcha, rutinas de prueba y error de ejecución.
- **Memoria de programa:** Destinada a almacenar la secuencia a realizar a partir de las señales de entrada, así como los datos de configuración del PLC.
- **Memoria de Datos y Memoria Imagen E/S:** Memoria encargada de almacenar los datos resultantes de cálculos y del uso de variables internas, ligada con la ALU (Unidad Aritmética-Lógica), así como también almacena los últimos estados de las entradas o las enviadas por las salidas.
- **Interfaz de Entrada y Salida:** Interfaces destinadas a comunicar el PLC con el entorno, recibiendo y enviando respectivamente información o instrucciones, que después serán comparados en el CPU, donde se realizarán acciones contempladas en el programa.

- **Fuente de Alimentación:** Unidad donde, a través de tensión exterior, se le provee la energía necesaria al PLC para su funcionamiento.
- **Buses de Comunicación:** Conexiones que permiten la comunicación entre las unidades de memoria, la CPU, las interfaces de salida y entrada, contamos con 3 buses:
  - Bus de control: Modera los intercambios de información
  - Bus de datos: Transfiere datos del sistema
  - Bus de dirección: Direccionamiento de la memoria y de los demás periféricos
- **Contadores:** Basados en los contadores digitales, pueden realizar el conteo de eventos externos, indicados a través de las entradas.
- **Temporizadores:** Actúa como un contador, con la diferencia que no realiza el conteo de eventos externos, lo hace a través de un generador de pulsos o de frecuencia dentro de la CPU.
- **Buses de Campo:** Unidades destinadas a permitir el intercambio de datos entre varios dispositivos, ya sean PLCs, PCs u otros, que puedan usar protocolos de información, ya sean Porfibus, Profinet, MPI, DeviceNet, IO Link, etc.
- **Conversores Analógico-Digitales:** Destinados a leer datos analógicos y convertirlos a datos binarios.

Clasificación de PLC.

El Plc se clasifica en compacto, modular, montaje en rack, con panel operador, ordenador industrial, de ranura, tipo software, de banda baja y de banda estrecha.

El termino PLC, en realidad se corresponde con una sigla en inglés, que hacen alusión a un controlador lógico programable, el cual hace referencia a un aparato electrónico, que cuenta con memoria programable, en la cual se le configuran una serie de funciones, que debe de llevar a cabo por medio de la activación de comandos.

Tipos de PLC.

- Compactos. Aquellos que están conformados por CPUS, PS, y módulos de entrada y salida en un mismo compartimiento, disponen por igual, de una entrada donde se puede medir la alta velocidad, y a la par cuenta con dos controladores analógicos.
- Modular. Son más variados que los compactos, ya que estos presentan el CPU, en un compartimiento aparte al SM y el CP, y aunque disponen de poco espacio para la distribución o colocación de módulos, existe la posibilidad de amplitud de estos. Estos pueden soportar una gran variedad de entradas y salidas, aunado a ello cuenta con una memoria más amplia, por lo que pueden albergar un programa más complejo, almacenar una cantidad moderada de datos y pueden realizar o enviar respuestas.
- Montaje en rack. En este caso los módulos no se presentan almacenados todos en un mismo compartimiento o bien se encuentran segmentados, sino que se dispone de un modo organizado

en el panel frontal del PLC, algunos expertos consideran que estos pueden brindar una respuesta más rápida a los comandos, dado que permiten un intercambio de datos a mayor velocidad.

- Con panel operador. Cuenta con un interfaz que facilita y optimiza su funcionamiento, y que brindan una supervisión constante y actividad de monitoreo a las actividades que se presentan en las maquinas; especial mención merece la interfaz ya que esta se presenta con una pantalla y teclas que facilitan la introducción de comandos y por ende la generación de respuesta.
- Ordenador Industrial. De peculiar conformación, ya que presentan dos PLC, uno que se haya en un pc y el otro en un compartimiento, es posible también que una de estos se encuentre en un área de hardware, mientras que otro se ubique en un espacio con software virtual.
- De ranura. De especial conformación, ya que esta trata de una tarjeta, como todas las que se emplean en el área de la informática, por medio de la cual se transmiten las funciones con mayor facilidad, esto hace que el Plc, sea más versátil pudiendo la misma liviana, ya que consta de una ranura desde la cual se puede controlar la tarjeta.
- Tipo Software. Este es el más moderno de todos, ya que trata en sí de un Plc virtual, es decir, diseñado para que pueda ser adaptado en cualquier ordenador o dispositivo, con la finalidad de que pueda ser empleado o bien monitoreado desde cualquier especial, y llevar a cabo sus funciones con mayor resguardo, por lo que este se trata de un uso más sencillo.
- Banda Baja. Puede trabajar a gran velocidad, como también facilita un campo de frecuencia mucho más amplio, por lo que fácilmente puede monitorear mayor cantidad de señales que pueda percibir.
- Banda Estrecha. Se trata de aquellos que pueden monitorear una frecuencia mínima, es decir, que pueden trabajar o bien maniobrar a un rendimiento inferior, por lo que resulta ideal para el monitoreo de tareas caseras o bien domésticas, más no industriales.

Bibliografía.

Bustamante, José. (2016). Curso PLC y Programación. Todo sobre PLC. Primera edición. Editorial: CreateSpace Independent Publishing Platform. España. ISBN: 978-1530245116

Microsoft Word - monografiaplc.doc (utb.edu.co)

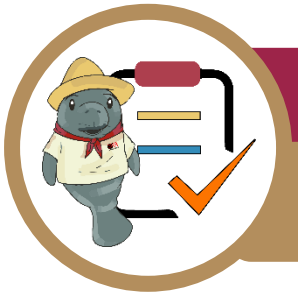
Microsoft PowerPoint - PLC's.ppt (uniovi.es)

Vallejo, Horacio Daniel. Paquete Coleccionable Todo Sobre Autómatas Programables No. 54 de la Serie Soluciones Prácticas. Editorial Quark. ISBN: 9789876231343

Actividad 1. Características de los PLC

Instrucciones: Llena la siguiente tabla comparativa donde incluyas cinco tipos de PLC (puedes elegir entre compacto, modular, de rack, tipo software, etc.). Toma en consideración la lectura anterior y realiza una investigación bibliográfica si lo consideras necesario.

No.	Nombre del tipo de PLC	Características principales	Ventajas	Limitaciones	Ejemplo de aplicación real
1					
2					
3					
4					
5					



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 1

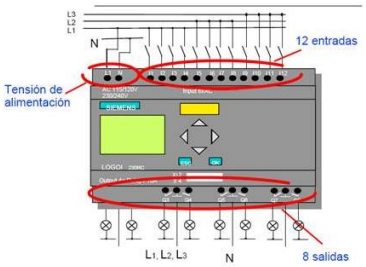
Actividad 1: Características de los PLC

DATOS GENERALES						
Nombre(s) del alumno (s)				**Matriculas(s)**		
Producto: Tabla				**Fecha**		
Formación laboral básica: Robótica **Submódulo 2:** Control industrial				**Periodo: 2026-2027A**		
Nombre del docente:				**Firma del docente**		
No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Incluye los cinco tipos diferentes de PLC.			2		
2.	Se describen claramente las características principales de cada tipo de PLC.			2		
3.	Se identifican al menos una ventaja y una limitación por cada tipo de PLC.			2		
4.	Se proporciona un ejemplo de aplicación real para cada tipo de PLC.			2		
5	La información está organizada en forma clara y comprensible.			2		
				CALIFICACIÓN		

Lectura 2. Puertos de entrada y salida

Puertos de entrada y salida.

Una de las clasificaciones más comunes de los PLC hace referencia en forma directa a la cantidad de entradas y salidas (E/S o I/O) de un PLC y nos dice que un PLC es considerado micro PLC cuando tienen menos de 64 E/S, pequeños cuando tienen menos de 256 E/S, medianos cuando tienen menos de 1024 E/S y grandes cuando tienen más de 1024 E/S.



Dispositivos de entrada.

Los dispositivos de entrada y salida son aquellos equipos que intercambian (o envían) señales con el PLC. Cada dispositivo de entrada es utilizado para conocer una condición de su entorno, como temperatura, presión, posición, entre otras. Entre estos dispositivos están: Sensores inductivos magnéticos, ópticos, pulsadores, termocuplas, termoresistencias, encoders, etc.

Dispositivos de salida.

Los dispositivos de salida son aquellos que responden a las señales que reciben del PLC, cambiando o modificando su entorno. Entre los dispositivos típicos de salida podemos hallar: Contactores de motor, electroválvulas, indicadores luminosos o simples relés.

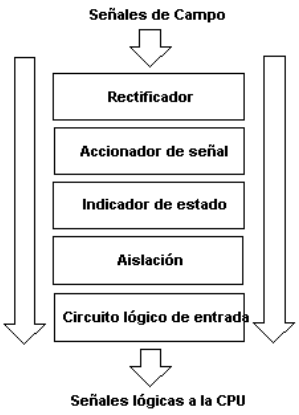
Generalmente los dispositivos de entrada, los de salida y el microprocesador trabajan en diferentes niveles de tensión y corriente. En este caso las señales que entran y salen del PLC deben ser acondicionadas a las tensiones y corrientes que maneja el microprocesador, para que éste las pueda reconocer. Ésta es la tarea de las interfases o módulos de entrada o salida.

Clasificación de las entradas.

Entradas Digitales: también llamadas binarias u "on-off", son las que pueden tomar sólo dos estados: encendido o apagado, estado lógico 1 ó 0. Los módulos de entradas digitales trabajan con señales de tensión. Cuando por un borne de entrada llega tensión, se interpreta como "1" y cuando llega cero tensión se interpreta como "0". Existen módulos o interfases de entradas de corriente continua para tensiones de 5, 12, 24 ó 48 Vcc y otros para tensión de110 ó 220 Vca. Los PLC modernos tienen módulos de entrada que permiten conectar dispositivos con salida PNP o NPN en forma indistinta. La diferencia entre dispositivos

con salida PNP o NPN es como la carga (en este caso la carga es la entrada del PLC) está conectada con respecto al neutro o al positivo.

Las señales digitales en contraste con las señales analógicas no varían en forma continua, sino que cambian en pasos o en incrementos discretos en su rango. La mayoría de las señales digitales utilizan códigos binarios o de dos estados. Las entradas discretas, tanto las de la corriente continua como las de la corriente alterna, están compuestas por una estructura típica que se puede separar en varios bloques:



- Rectificador: en el caso de una entrada de corriente alterna, convierte la señal en continua. En el caso de una señal de corriente continua, impide daños por inversión de polaridad.
- Acondicionador de señal: elimina los ruidos eléctricos, detecta los niveles de señal para los cuales conmuta el estado lógico, y lleva la tensión al nivel manejado por la CPU.
- Indicador de estado: en la mayoría de los PLC existe un indicador luminoso por cada entrada. Este indicador (casi siempre un LED) se encenderá con la presencia de tensión en la entrada y se apagará en caso contrario.
- Aislación: en la mayoría de los PLC las entradas se encuentran aisladas para que, en caso de sobretensiones externas, el daño causado no afecte más que a esa entrada, sin perjudicar el resto del PLC.
- Circuito lógico de entrada: es el encargado de informar a la CPU el estado de la entrada cuando éste lo interrogue.

Cuando la señal llega hasta los bornes del PLC tiene que atravesar todos estos bloques.

Recorrer este camino le lleva un tiempo que es llamado: tiempo de respuesta de la entrada.

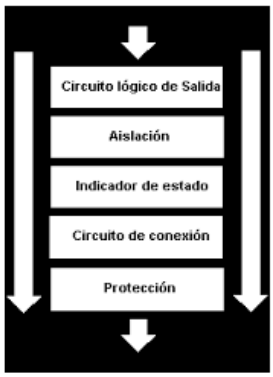
Un aspecto a analizar es el mínimo tiempo de permanencia o ausencia de una señal requerida para que el PLC la interprete como 0 ó 1. Si una variable de proceso pasa al estado lógico 1, y retorna al estado 0 en un tiempo inferior al tiempo de respuesta de la entrada, es posible que le PLC no llegue a leerla.

Entradas Analógicas: estos módulos o interfases admiten como señal de entrada valores de tensión o corriente intermedios dentro de un rango, que puede ser de 4-20 mA, 0-5 VDC o 0-10 VDC, convirtiéndola en un número. Este número es guardado en una posición de la memoria del PLC. Los módulos de entradas analógicas son los encargados de traducir una señal de tensión o corriente proveniente de un sensor de temperatura, velocidad, aceleración, presión, posición, o cualquier otra magnitud física que se quiera medir en un número para que el PLC la pueda interpretar. En particular es el conversor analógico digital (A/D) el encargado de realizar esta tarea. Una entrada analógica con un conversor A/D de 8 bits podrá dividir el rango de la señal de entrada en 256 valores (28).

Una señal es analógica cuando las magnitudes de esta se representan mediante variables continuas, análogas (relación de semejanza entre cosas distintas) a las magnitudes que dan lugar a la generación de esta señal.

Salidas del PLC.

- Circuitos lógicos de salida: es el receptor de la información enviada por la CPU.
- Aislación: cumple la misma función que en las interfases de entrada.
- Indicador de estado: también tiene la misma función que en la entrada.
- Circuitos de conexión: está compuesto por el elemento de salida al campo que maneja la carga conectada por el usuario. Existen tres tipos de circuitos de conexión que se describirán más adelante.
- Protección: son internas al PLC y pueden ser fusibles en serie con los contactos de salida, alguna protección electrónica por sobrecarga, o algún circuito RC. Recordar que en caso de que más de una salida use un solo borne de referencia, es éste el que lleva asociada la protección. Por lo cual si esta protección actúa dejarán de funcionar todas las salidas asociadas a ese borne común.



Tiempo de respuesta de la salida: al igual que en las entradas, se denomina tiempo de respuesta de la salida al tiempo que tarda una señal para pasar por todos los bloques. Existen cuatro posibilidades para el circuito de conexión de una salida:

Salida a relé: Es una de las más usuales. Con ellos es posible conectar tanto cargas de corriente alterna como continua. Suelen soportar hasta 2A de corriente. Una buena práctica en la instalación es verificar que la corriente máxima que consume la carga esté dentro de las especificaciones de la salida del PLC. Los tiempos de conmutación de estos tipos de salidas llegan a los 10 mseg. tanto para la conexión como para la desconexión. Algunas cargas son muy problemáticas, por ejemplo, las cargas inductivas, que tienen la tendencia a devolver corriente al circuito cuando son conectadas. Siendo la corriente estimada en unas 30 veces a la corriente de consumo nominal. Esto genera picos de voltaje que pueden dañar la salida a la que está conectada la carga. Para minimizar estos riesgos se utilizan comúnmente diodos, varistores u otros circuitos de protección. Los relés son internos al PLC. El circuito típico es el que se muestra en la figura de arriba. Cuando el programa active una salida, el PLC aplicará internamente tensión a la bobina del relé. Esta tensión hará que se cierren los contactos de dicho relé. En ese momento una corriente externa pasará a través de esos contactos y así se alimentará la carga. Cuando el programa desactiva una salida, el PLC desactiva la bobina abriendo así los contactos.

Salidas a transistor: Sólo son capaces de operar con corriente continua, de baja potencia (hasta 0,5 A) Pero tienen tiempos de conmutación que rondan el milisegundo y una vida útil mucho mayor que la de los relés. En este tipo de salida el transistor es el encargado de

conectar la carga externa cuando el programa lo indique.

Salidas por triac: Manejan corrientes alternas. Al igual que los transistores, por ser semiconductores tienen una vida útil mucho mayor que la del relé, que es un elemento electromecánico.

Salidas analógicas: Los módulos de salida analógica permiten que el valor de una variable numérica interna del autómata se convierta en tensión o corriente.

Internamente en el PLC se realiza una conversión digital analógica (D/A), puesto que el autómata sólo trabaja con señales digitales. Esta conversión se realiza con una precisión o resolución determinada (número de bits) y en un intervalo determinado de tiempo (período muestreo). Esta tensión o intensidad puede servir de referencia de mando para actuadores que admitan mando analógico, como pueden ser las válvulas proporcionales, los variadores de velocidad, las etapas de los tiristores de los hornos, los reguladores de temperatura, etc. Permitiendo al autómata realizar funciones de regulación y control de procesos continuos.

Bibliografía.

Bustamante, José. (2016). Curso PLC y Programación. Todo sobre PLC. Primera edición. Editorial: CreateSpace Independent Publishing Platform. España. ISBN: 978-1530245116

Microsoft Word - monografiaplc.doc (utb.edu.co)

Microsoft PowerPoint - PLC's.ppt (uniovi.es)

Vallejo, Horacio Daniel. Paquete Coleccionable Todo Sobre Autómatas Programables No. 54 de la Serie Soluciones Prácticas. Editorial Quark. ISBN: 9789876231343

Actividad 2. Puertos de entrada y salida PLC

Instrucciones: Completa la actividad cumpliendo con cada uno de los puntos que se enlistan.

1. Lee la siguiente lista de dispositivos que pueden formar parte de un sistema automatizado.
2. Investiga sobre cada uno de ellos.
3. Clasifica cada dispositivo en la tabla, indicando: Si es dispositivo de entrada o de salida y si usa señal digital o analógica.

Lista de dispositivos:

- Sensor inductivo: _____
  _____
  _____
- Termocupla: _____
  _____
  _____
- Indicador luminoso: _____
  _____
  _____
- Electroválvula: _____
  _____
  _____
- Encoder: _____
  _____
  _____
- Válvula proporcional: _____
  _____
  _____
- Pulsador: _____

_____
_____

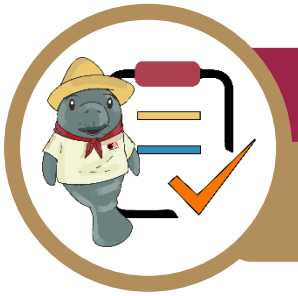
- Variador de velocidad: _____
_____
_____

- Relé: _____
_____
_____

- Sensor de presión (con señal 4-20 mA): _____
_____
_____

Clasificación de dispositivos

Dispositivo	Entrada/Salida	Tipo de señal (Digital/Analógica)
Sensor inductivo		
Termocupla		
Indicador luminoso		
Electroválvula		
Encoder		
Válvula proporcional		
Pulsador		
Variador de velocidad		
Relé		
Sensor de presión (con señal 4-20 mA)		



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 2

Actividad 2: Puertos de entrada y salida PLC

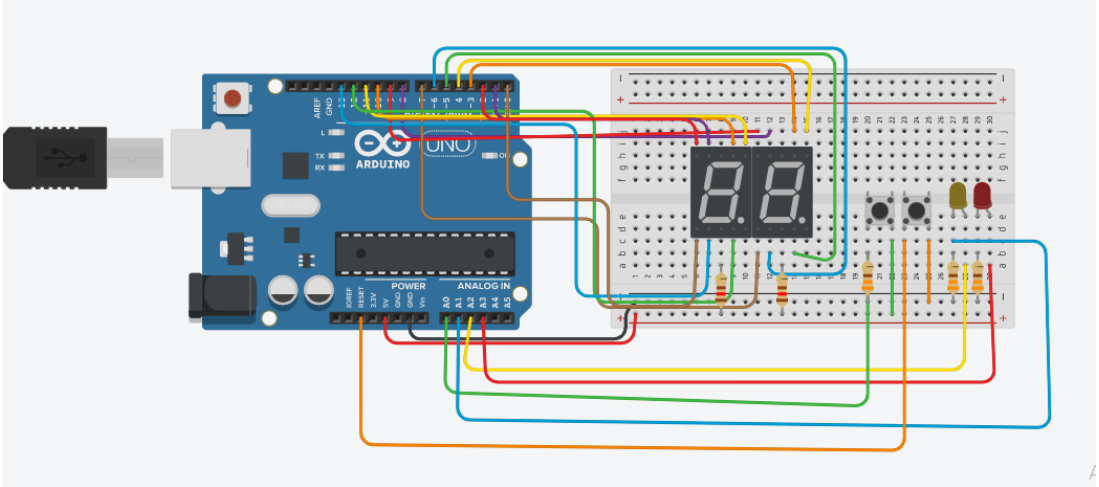
DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Tabla		**Fecha**	
Formación laboral básica: Robótica **Submódulo 2:** Control industrial		**Periodo: 2026-2027ª**	
Nombre del docente:		**Firma del docente**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Clasificó correctamente si el dispositivo es de entrada o salida en al menos 8 de los 10 casos.			4		
2.	Identificó correctamente el tipo de señal (digital o analógica) en al menos 8 de los 10 casos.			4		
3.	La redacción de las investigaciones es clara, coherente y refleja comprensión del funcionamiento.			1		
4.	La presentación es ordenada, limpia y con formato adecuado (uso correcto de la tabla, sin errores).			1		
				CALIFICACIÓN		

Ejercicio 1. Proyecto PLC en Tinkercad

Instrucciones: Realiza el siguiente ejercicio de PLC en Tinkercad.

Circuito para realizar.



Componentes.

- 1 placa de pruebas pequeña.
- Arduino Uno R3.
- 1 led rojo.
- 1 led amarillo.
- 2 pulsadores.
- 3 resistencias de 330 Ohm.
- 2 resistencias de 120 Ohm.
- 1 visualizador de 7 segmentos.
- Cables: azul, rojo amarillo, verde, naranja.

Código.

```
int arr=A0;
int dato=A1;
int i=A2;
int d=A3;
int val=0;
int par=7;
int val2=0;
int val3=0;

void setup()
{
  pinMode(d, OUTPUT);
  pinMode(i, OUTPUT);
  pinMode(arr, INPUT);
  pinMode(par, INPUT);
  pinMode(dato, INPUT);

  pinMode(3, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
```

```
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
  pinMode(0, OUTPUT);
  pinMode(1, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
}

void loop()
{

   val = digitalRead(arr);
 val2 = digitalRead(par);
  val3 = digitalRead(dato);

    if(val == HIGH && val3 ==
LOW) {

  digitalWrite(d, HIGH);
   //0//
  digitalWrite(10, LOW);
  digitalWrite(11, LOW);
  digitalWrite(12, LOW);
  digitalWrite(13, LOW);
  digitalWrite(0, LOW);
  digitalWrite(1, LOW);
  digitalWrite(2, HIGH);

  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
  digitalWrite(8, LOW);
  digitalWrite(9, HIGH);
  delay(1000);

  digitalWrite(3, HIGH);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, HIGH);
  digitalWrite(7, HIGH);
  digitalWrite(8, HIGH);
  digitalWrite(9, HIGH);
  delay(1000);

  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, HIGH);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
  digitalWrite(8, HIGH);
  digitalWrite(9, LOW);
  delay(1000);

  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, HIGH);
  digitalWrite(8, HIGH);
  digitalWrite(9, LOW);
  delay(1000);

  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, HIGH);
  digitalWrite(7, HIGH);
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
  delay(1000);

  digitalWrite(3, HIGH);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, HIGH);
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
  delay(1000);

  digitalWrite(3, LOW);
  digitalWrite(4, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, HIGH);
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
  delay(1000);

  digitalWrite(3, LOW);
  digitalWrite(4, HIGH);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
  digitalWrite(8, LOW);
  digitalWrite(9, HIGH);
  delay(1000);

  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, HIGH);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
  digitalWrite(8, HIGH);
  digitalWrite(9, LOW);
  delay(1000);

  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, HIGH);
  digitalWrite(7, HIGH);
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
  delay(1000);

  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, HIGH);
  digitalWrite(8, HIGH);
  digitalWrite(9, LOW);
  delay(1000);

  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
  digitalWrite(8, LOW);
  digitalWrite(9, HIGH);
  delay(1000);

  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, HIGH);
  digitalWrite(7, HIGH);
  digitalWrite(8, HIGH);
  digitalWrite(9, LOW);
  delay(1000);

  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
  digitalWrite(8, LOW);
  digitalWrite(9, HIGH);
  delay(1000);
   //1//
  digitalWrite(10, HIGH);
  digitalWrite(11, LOW);
  digitalWrite(12, LOW);
  digitalWrite(13, HIGH);
  digitalWrite(0, HIGH);
  digitalWrite(1, HIGH);
  digitalWrite(2, HIGH);

  digitalWrite(3, LOW);
  digitalWrite(4, LOW);
  digitalWrite(5, LOW);
  digitalWrite(6, LOW);
  digitalWrite(7, LOW);
  digitalWrite(8, LOW);
  digitalWrite(9, HIGH);
  delay(1000);
```

```
digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, HIGH);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);
```

```
digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);


digitalWrite(d, LOW);


//2//
digitalWrite(10, LOW);
digitalWrite(11, LOW);
digitalWrite(12, HIGH);
digitalWrite(13, LOW);
digitalWrite(0, LOW);
```

```
digitalWrite(1, HIGH);
digitalWrite(2, LOW);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, HIGH);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
```

```
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

//3//
digitalWrite(10, LOW);
digitalWrite(11, LOW);
digitalWrite(12, LOW);
digitalWrite(13, LOW);
digitalWrite(0, HIGH);
digitalWrite(1, HIGH);
digitalWrite(2, LOW);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, HIGH);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(i, HIGH);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
```

```
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);


//4//
digitalWrite(10, HIGH);
digitalWrite(11, LOW);
digitalWrite(12, LOW);
digitalWrite(13, HIGH);
digitalWrite(0, HIGH);
digitalWrite(1, LOW);
digitalWrite(2, LOW);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, HIGH);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

//5//
digitalWrite(10, LOW);
digitalWrite(11, HIGH);
digitalWrite(12, LOW);
digitalWrite(13, LOW);
digitalWrite(0, HIGH);
digitalWrite(1, LOW);
digitalWrite(2, LOW);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);
```

```
digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, HIGH);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
```

```
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);


//6//
digitalWrite(10, LOW);
digitalWrite(11, HIGH);
digitalWrite(12, LOW);
digitalWrite(13, LOW);
digitalWrite(0, LOW);
digitalWrite(1, LOW);
digitalWrite(2, LOW);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, HIGH);
```

```
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, HIGH);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
```

```
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);


//7//
digitalWrite(10, LOW);
digitalWrite(11, LOW);
digitalWrite(12, LOW);
digitalWrite(13, HIGH);
digitalWrite(0, HIGH);
digitalWrite(1, HIGH);
digitalWrite(2, HIGH);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, HIGH);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);

digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);


//8//
digitalWrite(10, LOW);
digitalWrite(11, LOW);
```

```
digitalWrite(12, LOW);
digitalWrite(13, LOW);
digitalWrite(0, LOW);
digitalWrite(1, LOW);
digitalWrite(2, LOW);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, HIGH);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
```

```
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
```

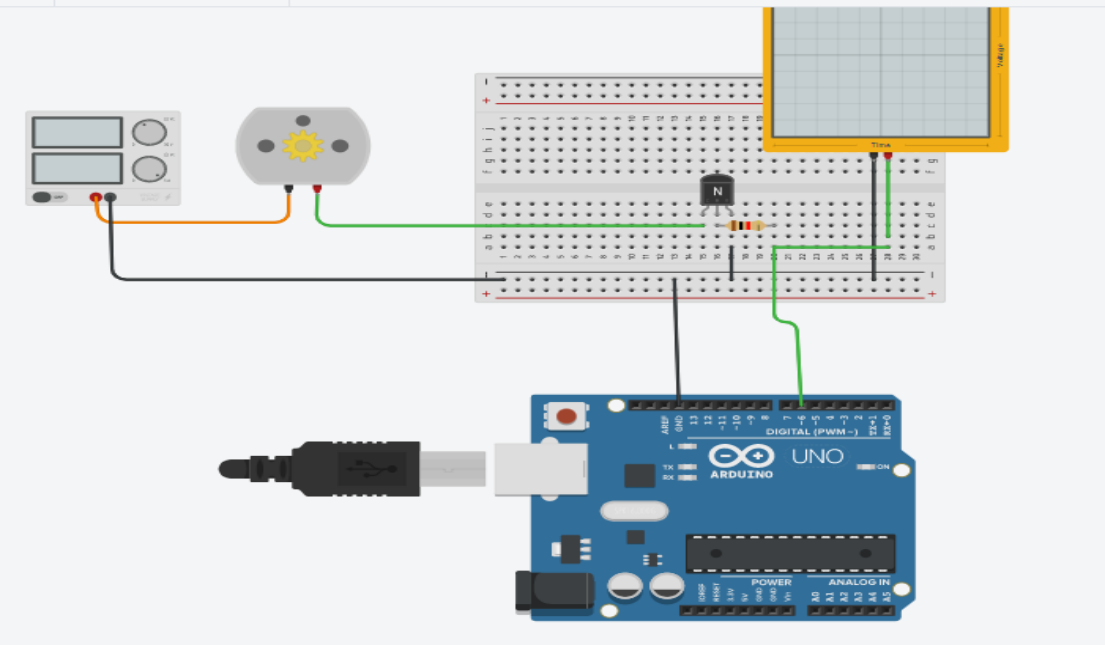
```
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

//9//
digitalWrite(10, LOW);
digitalWrite(11, LOW);
digitalWrite(12, LOW);
digitalWrite(13, LOW);
digitalWrite(0, HIGH);
digitalWrite(1, LOW);
digitalWrite(2, LOW);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, HIGH);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
```

```
digitalWrite(8, HIGH);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, HIGH);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, HIGH);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);


digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, LOW);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);

digitalWrite(3, LOW);
digitalWrite(4, LOW);
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(7, HIGH);
digitalWrite(8, LOW);
digitalWrite(9, LOW);
delay(1000);
  }
}
```

Práctica 1. Control de velocidad básico de un motor DC con PWM y Arduino

Instrucciones: Realiza la práctica de velocidad básico de un motor DC utilizando PWM y Arduino en Tinkercad.

Circuito.



Componentes.

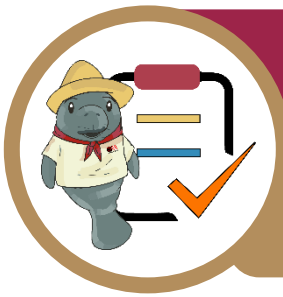
- 1 placa de pruebas pequeña.
- Arduino Uno R3.
- Suministro de energía de voltaje 12 y corriente 5.
- Osciloscopio de 0.5 ms.
- 1 transistor NPN.
- 1 resistencia 1 Kohm.
- 1 motor CC.
- Cables: rojo, verde, negro.

Código.

```
/*
Esta aplicación permite el control de la velocidad PWM de un motor DC de forma temporizada

Conexiones:

Pin 6 ---> Base de transistor NPN
*/

void setup()
{
   pinMode(6,OUTPUT);//Pin 6 como salida digital , por allí se genera la señal PWM
}

void loop()
{
analogWrite(6,0); //Motor apagado
delay(2000); //Pausa de 2 segundos
analogWrite(6,64); //Motor al 25% de la velocidad
delay(2000); //Pausa de 2 segundos
analogWrite(6,128);//Motor al 50% de la velocidad
delay(2000); //Pausa de 2 segundos
analogWrite(6,192);//Motor al 75% de la velocidad
delay(2000); //Pausa de 2 segundos
analogWrite(6,255);//Motor al 100% de la velocidad
delay(2000); //Pausa de 2 segundos
}
```

Conclusiones.

Analiza el código, así como el funcionamiento del circuito, elabora una conclusión en donde especifiques que te queda claro de la elaboración de este, la cual debe ser personal y entendible.
Contesta: ¿Qué aprendí? ¿cómo lo aplico en mi diario acontecer?



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 3

Práctica 1: Control de velocidad básico de un motor DC con PWM y Arduino

DATOS GENERALES						
Nombre(s) del alumno (s)				**Matriculas(s)**		
Producto: Reporte de práctica				**Fecha**		
Formación laboral básica: Robótica **Submódulo 2:** Control industrial				**Periodo: 2026-2027A**		
Nombre del docente:				**Firma del docente**		

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Ingresa a Tinkercad.			1		
2.	Realiza el circuito con los componentes indicados.			2		
3.	Realiza las conexiones con el color de cable especificado.			2		
4.	Realiza el código sin errores.			2		
5	Ejecuta el circuito y funciona en forma correcta			2		
6.	Elabora una conclusión personal y entendible.			1		
				CALIFICACIÓN		

Ejercicio 2. Simulación de circuito lógico en Tinkercad

Instrucciones: Realiza el Ejercicio 2 siguiendo los pasos que se especifican.

Circuito lógico.

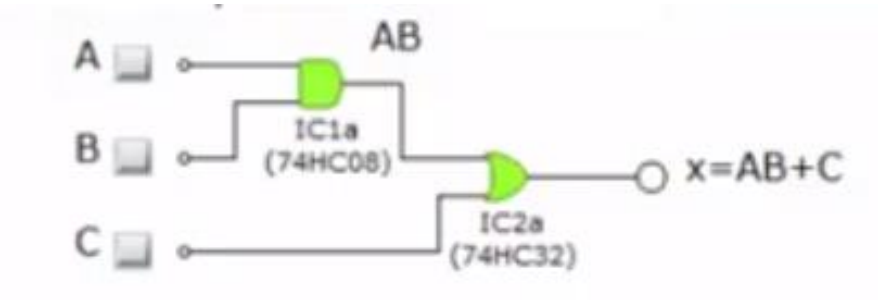
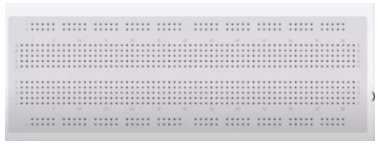


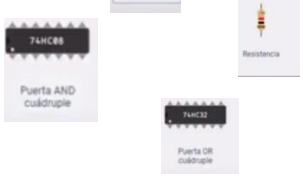
Tabla de verdad.

En la tabla se deben mostrar todas las posibles combinaciones de niveles lógicos presentes en las entradas A, B y C con la correspondiente salida en X.

ENTRADAS			SALIDA
A	B	C	X
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Componentes en Tinkercad.



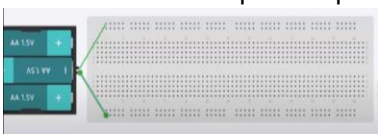


- Protoboard de prueba
- Set de 3 pilas de 1.5 V
- SPST de conmutadoras de 4 entradas
- 3 resistencias de 330 Ohm
- Puerta AND cuádruple

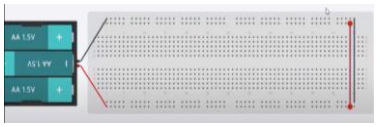
- Puerta OR cuádruple
- 1 resistencia de 300 Ohm
- 1 led rojo
- Cables en color rojo, negro y verde


Procedimiento.

- Conectar el set de pilas al protoboard del negativo a tierra y de positivo a corriente eléctrica.

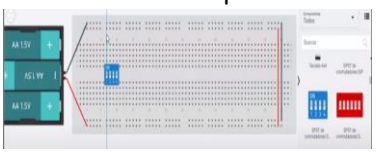


- Cambiar el color del cable de tierra a negro y el de corriente eléctrica a rojo.
- Realizar dos puentes en el protoboard en el externo derecho uno a tierra y el otro a corriente eléctrica con cable negro y rojo, como se muestra en la figura.

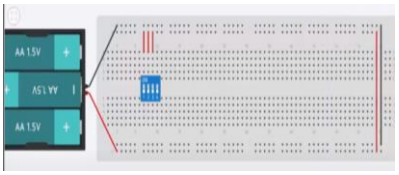


- Se revisa el circuito lógico para hacer las conexiones con los componentes necesarios.

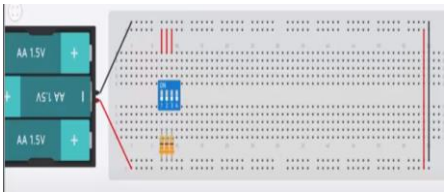
- Se coloca en el protoboard el componente SPST cuádruple como se indica en la imagen.

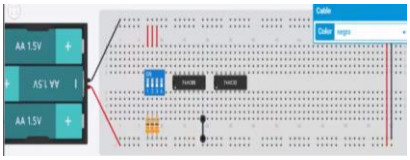
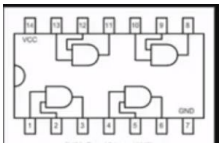
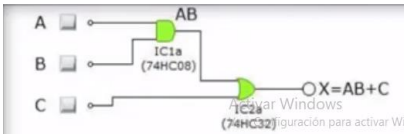


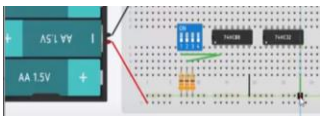
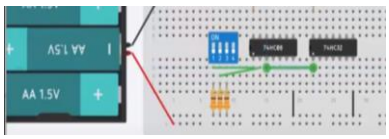
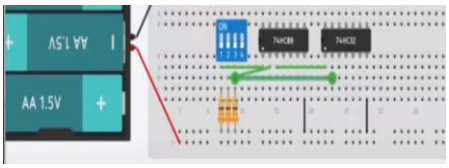
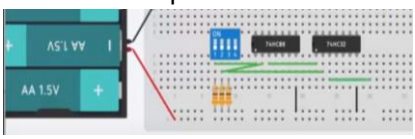
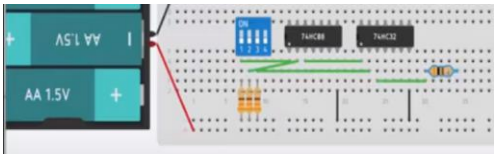
- Se realizan las conexiones del SPST a corriente eléctrica con cable color rojo como se indica.



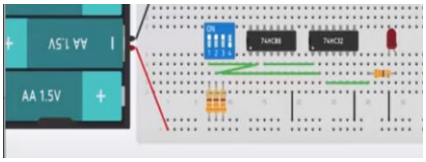
- Se colocan las 3 resistencias de 330 Ohm de la forma en que se indica.



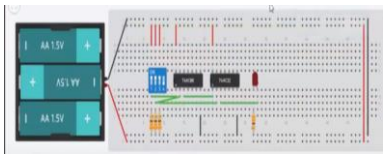
- Se coloca la puerta AND y OR cuádruple como se muestra en la imagen. 
- Se realiza la conexión a tierra de la puerta AND con cable negro a tierra como se indica. 
- Se muestra las conexiones con sus puertas lógicas de la puerta AND. 

- Se realiza las conexiones de SPST de la 1 y 2 a las entradas 1 y 2 de la puerta AND con cable verde. 
- Se conecta a tierra la puerta OR con cable color negro como se muestra.
- Se realiza una conexión con cable color verde de la entrada 3 de la puerta AND a la entrada 1 de la puerta OR. 
- De la entrada 2 de la puerta AND se conecta un cable color verde a la entrada 3 de SPST, de ahí se hace la conexión a la entrada 2 de la puerta OR, como se indica. 
- De la entrada 3 de la puerta OR se realiza una conexión con cable verde como se muestra en la imagen. 
- Se coloca la resistencia de 300 Ohm al cable verde que está conectado a la entrada 3 de la puerta OR como se muestra en la imagen. 

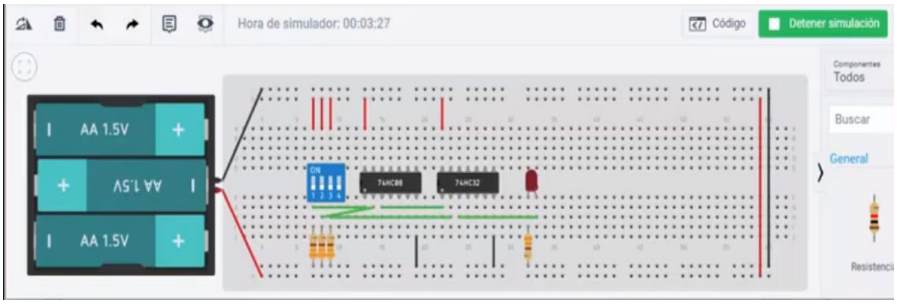
- Se coloca el led rojo conectado a tierra y a la resistencia de 300 Ohm como se muestra.



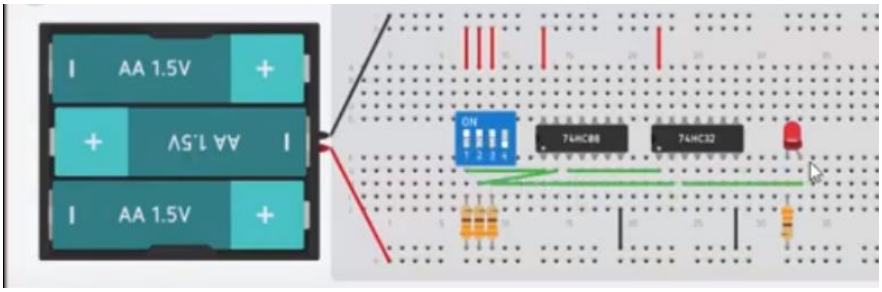
- Se hace la conexión de la puerta AND en la entrada 8 a corriente eléctrica con cable rojo, de la misma forma se hace la conexión de la entrada 8 de la puerta OR a corriente eléctrica con cable rojo.



- Se realiza la ejecución del circuito con las primeras entradas de A, B y C, en las cuales el valor es cero.
- Al estar las 3 entradas en 0 la salida X da como resultado:
- Obtener los valores de la tabla de verdad con la ejecución del circuito dependiendo de las entradas, lo cual se muestra con el encendido o (1) y el apagado o (0) del led rojo.
- El circuito se muestra de la siguiente forma cuando las 3 entradas están en 0.



- El circuito se muestra de la siguiente forma cuando las 3 entradas están en 1.



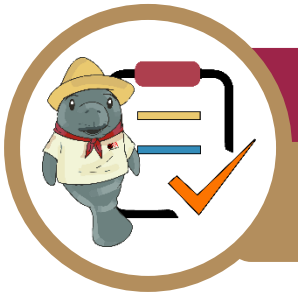
Bibliografía.

Álgebra de Boole.

Boylestad, Robert L, Nashelsky Louis. Electrónica: Teoría de Circuitos y Dispositivos Electrónicos. Editorial Pearson Educación de México, S. A.

Diversos videos para el diseño y construcción del circuito en el protoboard.

Todo sobre circuitos digitales y tablas de verdad.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 4

Ejercicio 2: Simulación de circuito lógico en Tinkercad

DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Reporte de práctica		**Fecha**	
Formación laboral básica: Robótica **Submódulo 2:** Control industrial		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Realiza las combinaciones de la tabla de verdad.			2		
2.	Realiza el circuito con los componentes indicados.			2		
3.	Realiza las conexiones con el color de cable especificado.			2		
4.	Coloca la compuerta AND y OR cuádruple como se indica.			2		
5	Ejecuta el circuito y funciona en forma correcta.			2		
				CALIFICACIÓN		

Lectura 3. Manejo, instalación y conexión en PLC

El término PLC proviene de las siglas en inglés para Programmable Logic Controler, que traducido al español se entiende como "Controlador Lógico Programable". Se trata de un equipo electrónico, que, tal como su mismo nombre lo indica, se ha diseñado para programar y controlar procesos secuenciales en tiempo real. Por lo general, es posible encontrar este tipo de equipos en ambientes industriales.

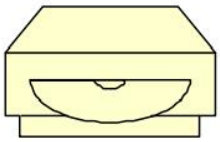
¿Cómo funciona un PLC?

a. El PLC está siempre repitiendo un ciclo, llamado ciclo de SCAN, que consiste en lo siguiente:
b. Lee todas las entradas y almacena el estado de cada una de ellas.
c. Ejecuta las operaciones del programa siguiendo el orden en que se han grabado.
d. En tercer lugar, escribe el resultado de las operaciones en las salidas.
e. Una vez escritas todas las salidas (activando o desactivando las que el resultado de las operaciones así lo requieran) vuelve al paso A.

Este ciclo de Scan se realiza indefinidamente hasta que pasemos el conmutador de la CPU a la posición STOP.

Partes que componen un control lógico programable.

Propios del PLC: Hardware es la parte física y software es el programa con el cual funciona el PLC.



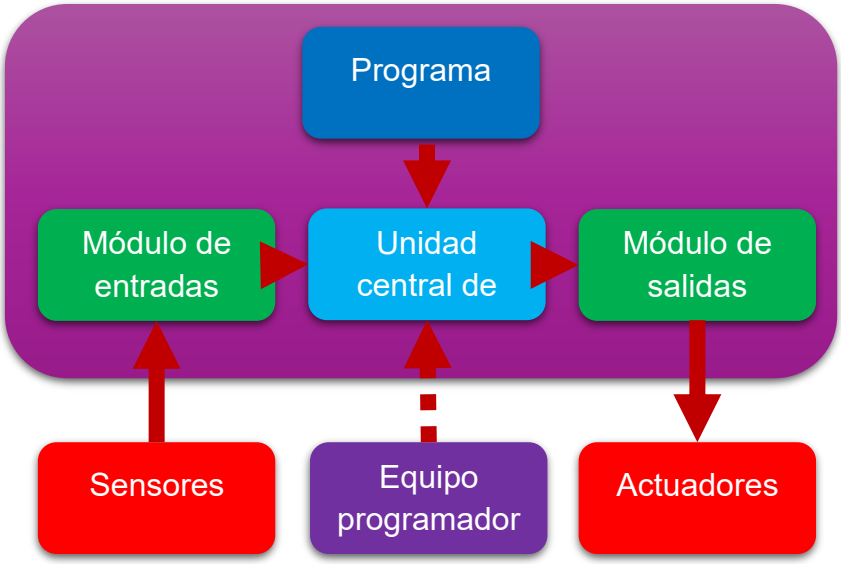
Externos del PLC: Sensores son elementos de entrada del PLC. Actuadores son elementos de salida del PLC. Equipo programador es el equipo con el cual se programa el PLC.

Simbología utilizada en el PLC.

Denominación	Símbolo
Contacto NA	--] [--
Contacto NC	--]/[--
Respuesta inmediata	--(=)--
Salida del PLC	--(S)--
Relay	--(R)--
Contador ascendente	--(CA)--
Contador descendente	--(CD)--
Temporizador	--(T)--

Características del montaje.

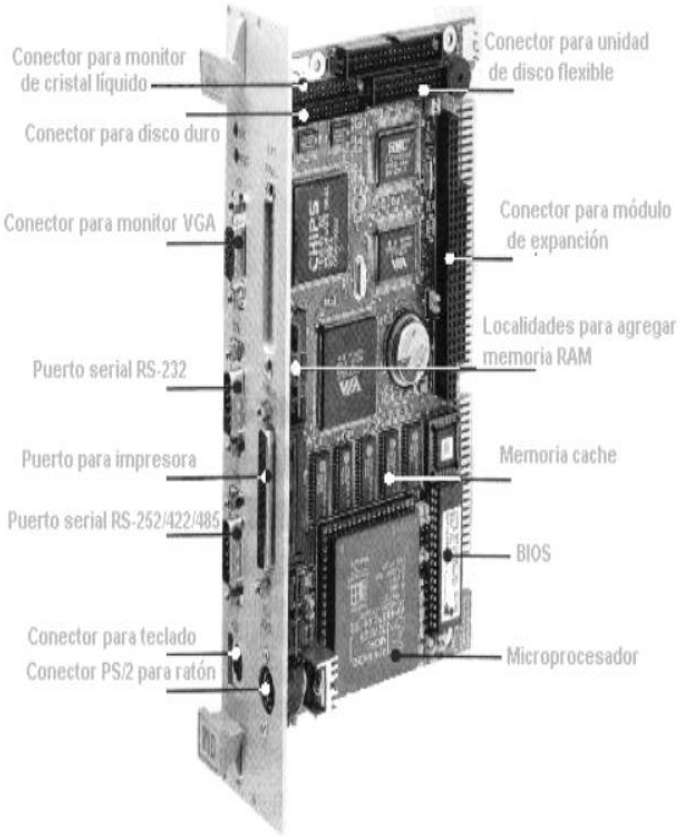
Para poder sabes las características de montaje de un PLC debemos tener en cuenta la arquitectura de este que a continuación se describe:

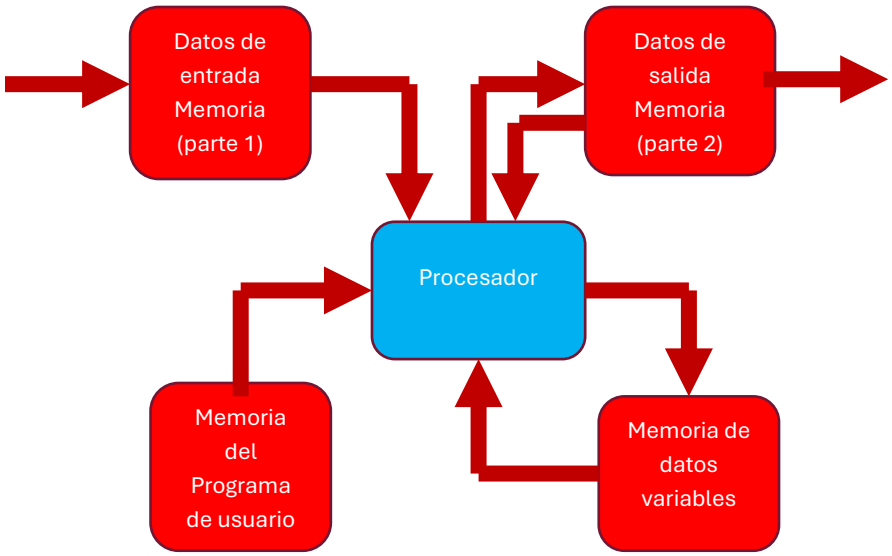


Partes que integran un control lógico programable.

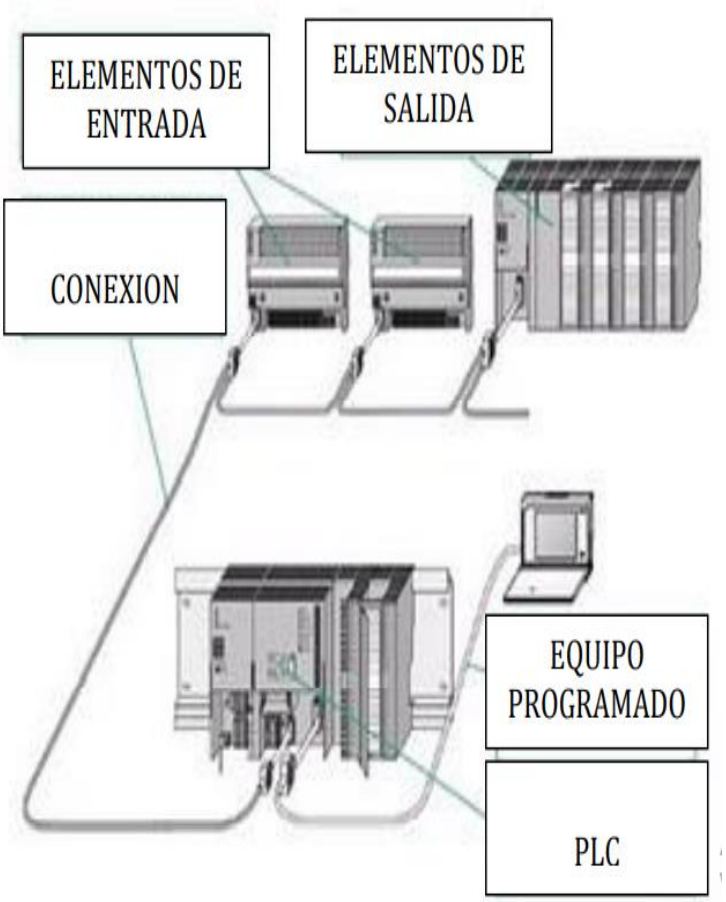
Las partes que integran a un Control Lógico Programable son las siguientes:

- Unidad central de proceso. Este bloque es el módulo principal, ya que tiene que realizar la gestión de ordenar y organizar la comunicación entre las distintas partes que conforman al PLC. Contiene y ejecuta el programa del usuario, que consiste en una serie de instrucciones que representa el proceso de control lógico que debe ejecutarse. Para poder hacer este trabajo, la unidad central de proceso debe almacenar las condiciones de entrada y salida más recientes.

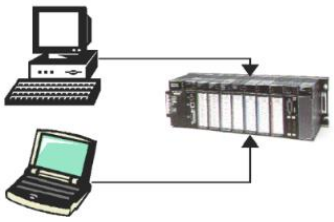




- Módulos de entrada y salida de datos. Se encargan del trabajo de intercomunicación entre los dispositivos exteriores al PLC y los circuitos electrónicos de baja potencia que conforman a la Unidad Central de Proceso. Están constituidos por tarjetas de circuitos impresos que contienen dispositivos capaces de aislar al PLC con el entorno exterior, además de contar con indicadores luminosos que informan de manera visual el estado que guardan las entradas y salidas. En los bornes de conexión de estos módulos de (E/S) están conectadas las señales de los sensores y actuadores, que vigilan y manipulan el proceso que sé está controlando. Los elementos de entrada pueden ser sensores o equipo manual de accionamiento como estación de botones etc. Los elementos de salida son los actuadores.



- Dispositivo de programación o terminal. Se trata de un elemento que aparentemente es complementario, pero se emplea con mucha frecuencia en la operación de un PLC, ya que es un dispositivo por medio del cual se van accesando las instrucciones que componen al programa de usuario que realiza las acciones de control industrial. Algunos PLC están equipados con un dispositivo de programación que físicamente tiene el aspecto de una calculadora, y en su teclado se encuentran todos los símbolos que se emplean para la elaboración de un programa de control, además cuenta también con una pantalla de cristal líquido en el que se exhibe gráficamente la representación de la tecla que fue oprimida. Normalmente el dispositivo programador se encuentra dedicado exclusivamente a la tarea de generar los comandos e introducirlos al PLC (acto de programar), este elemento por obvias razones es construido por la misma compañía que fabrica el PLC, por lo cual tiene que ser el adecuado y poseer toda la capacidad de comunicar al usuario con el PLC. El dispositivo programador requiere de un cable por medio del cual se envían las instrucciones del programa a la memoria de usuario del PLC, el cable que casi todos los fabricantes de PLC emplean conduce los datos en una comunicación serial. De acuerdo con la evolución que día con día se va obteniendo en el ramo de la electrónica, se generó otra manera de programar un PLC de forma más versátil, y es por medio del empleo de una computadora de escritorio o portátil, la cual necesariamente debe de contar en una de sus ranuras de expansión con una tarjeta de interfaz de comunicación. A través de un cable de comunicación serial se interconecta la tarjeta de interfaz con el microcontrolador del PLC, y por medio de un software especial que a la vez resulta amigable al usuario se va escribiendo el programa de control, para su posterior interpretación y envío al PLC.

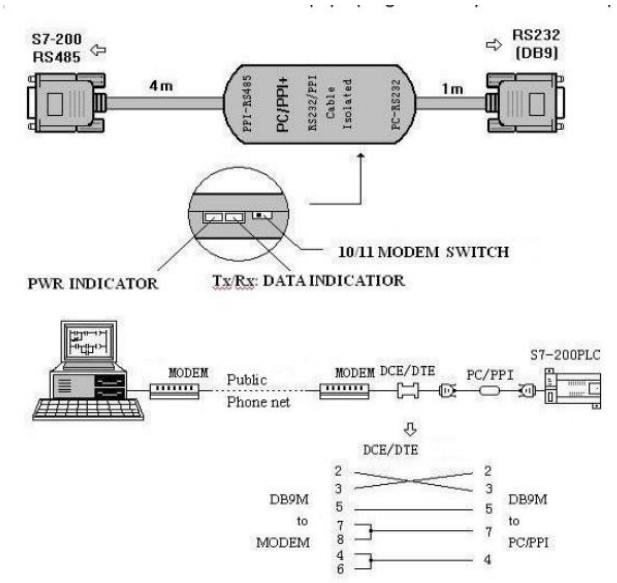


El empleo de una computadora personal cada vez cobra más auge ya que es muy fácil realizar la programación de un PLC, y en la actualidad no solo se genera el programa, sino que también se puede simular antes de que se descargue el programa en la memoria del PLC, fomentando con esto una mayor productividad y un mejor desempeño al prácticamente eliminar los posibles errores tanto de sintaxis como el error lógico.

Conexiones de la interfaz.

La conexión entre el equipo programador y el PLC es crucial ya que para poder programar al PLC se necesita de este medio al cual denominamos interfaz. La interfaz es la conexión entre el equipo programador y el PLC este es por medio de un cable de conexión el cual carga el programa que se realiza en la computadora

o equipo ordenador para después cargarlo a PLC y que este empiece a funcionar en los procesos que se requiera en la industria.



Bibliografía.

Bustamante, José. (2016). Curso PLC y Programación. Todo sobre PLC. Primera edición. Editorial: CreateSpace Independent Publishing Platform. España. ISBN: 978-1530245116

https://www.ipn.mx/assets/files/cecyt11/docs/Guias/UATecnologicas/IME/6toSemestre/

Microsoft Word - monografiaplc.doc (utb.edu.co)

Microsoft PowerPoint - PLC's.ppt (uniovi.es)

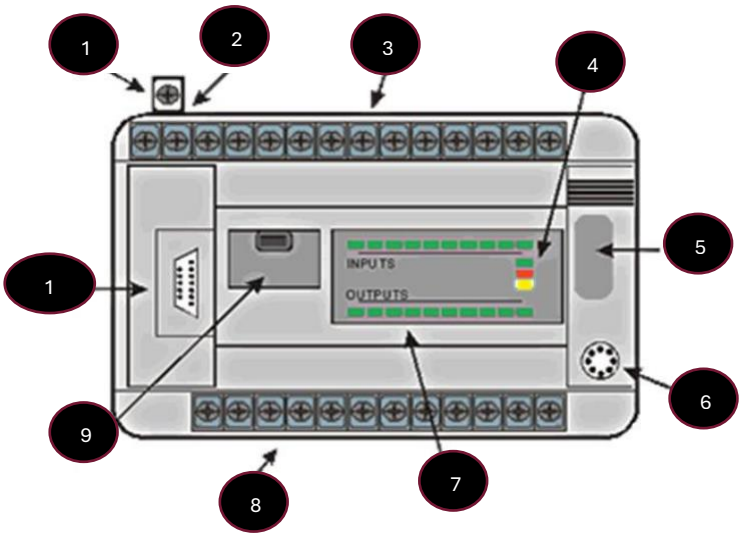
Vallejo, Horacio Daniel. Paquete Coleccionable Todo Sobre Autómatas Programables No. 54 de la Serie Soluciones Prácticas. Editorial Quark. ISBN: 9789876231343

Actividad 3. Conexiones, instalación y manejo de PLC

Instrucciones: Después de realizar la Lectura 3 Identifica las partes de los siguientes PLC, a que tipo pertenecen y anota el número correspondiente en cada componente.

Tipos de PLC: Modular, Compacto

PLC: _____Compacto_____



( ) Puerto de comunicación

( ) Terminales de conexión de entradas

( ) Batería

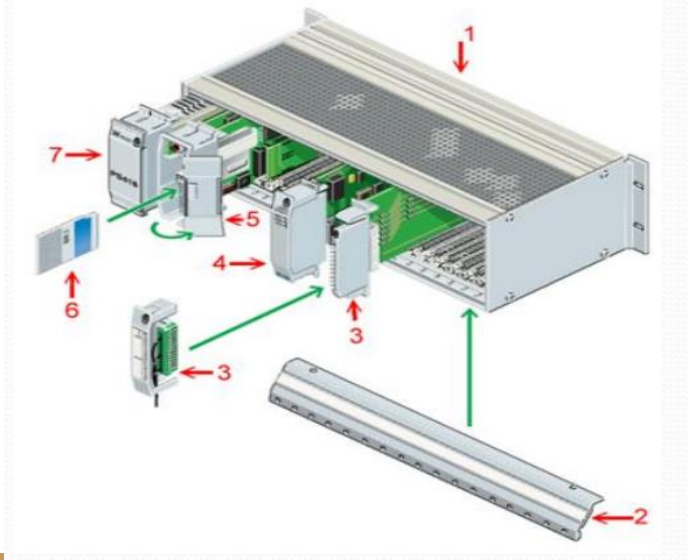
( ) Terminales de alimentación

( ) Terminales de conexión de salidas

( ) Tierra

( ) Panel de LEDS indicadores del estado de salidas y entradas

PLC: _____Modular_____



( ) Tarjeta de fuente de alimentación

( ) Tarjetas de entradas y salidas

( ) CPU

( ) Barra de compensación de potencial

( ) Tarjeta de memoria

( ) Rack

( ) Tarjetas de comunicación

Lectura 4. Programación básica lenguaje escalera

Para programar un PLC es necesario emplear un lenguaje específico, el cual en forma general sólo entiende éste. El lenguaje de programación de cada PLC cambia de acuerdo con el creador del producto y a pesar de que se utilizan los mismos símbolos en los distintos lenguajes, la forma en cómo se crean y almacenan cambia acorde al fabricante, debido a esto la forma en que se interpretan las instrucciones en un PLC depende de la marca. Existen comercialmente tres lenguajes que la mayoría de los fabricantes de PLC ponen a disposición de los usuarios, estos lenguajes son:

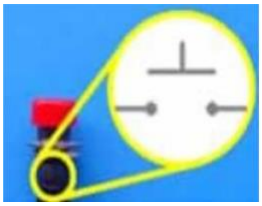
- Diagrama de Contactos, también conocido como Lenguaje en Escalera.
- Listado de instrucciones.
- Diagrama de funciones.

Diagrama de escalera.

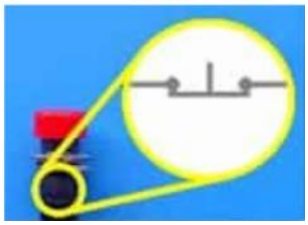
La mayoría de los fabricantes incorporan este lenguaje, debido a la semejanza con los esquemas de relés utilizados en los automatismos eléctricos de lógica cableada, lo que facilita la labor de los técnicos para trabajar con dicho automatismo. En este lenguaje, la tarea que realizar el autómata se representa gráficamente mediante un esquema de contacto. Para programar en este lenguaje se necesita una unidad de programación que posea una terminal con una pantalla semigráfica que permite visualizar el esquema de contactos.

Este lenguaje es una representación gráfica que por medio de software se implementan tanto los contactos físicos que posee un relé (variables de entrada), así como también las bobinas (variables de salida) que lo constituyen, las actividades que realizan estas representaciones se materializan por medio de las líneas de entrada y salida del PLC. En el lenguaje en Escalera son muy bastos los símbolos empleados, los elementos básicos correspondientes a las entradas, son los que a continuación se muestran:

- Contacto normalmente abierto (NA). Tiene la misma función de un botón real, el cual cuando no es accionado se reposiciona automáticamente a su estado natural que es encontrarse abierto o desconectado. Cuando el usuario presiona el interruptor hace que exista una unión entre los dos contactos internos que tiene el botón, cambiando su estado lógico de abierto (desconectado) a cerrado (conectado).



- Contacto normalmente cerrado (NC). Funciona como un botón real, pero de forma inversa al contacto normalmente abierto, cuando se acciona se reposiciona automáticamente a su estado natural que es el encontrarse cerrado o conectado.



De acuerdo con la convención establecida por los fabricantes de los PLC, la correspondencia que tienen los estados lógicos cerrado y abierto con los dígitos binarios "0" y "1" es la siguiente:

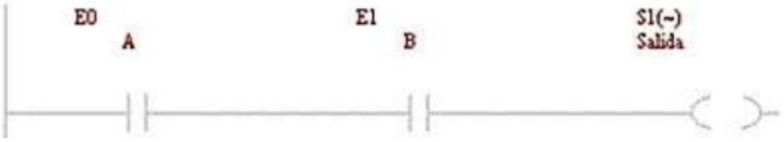
- Abierto equivale a "0" lógico.
- Cerrado equivale a "1" lógico.

Al conocer los símbolos básicos correspondientes a las entradas en el Lenguaje en Escalera, se debe encontrar la forma de obtener una respuesta con base a nuestras entradas. La solución la hallamos en el mismo Lenguaje Escalera, ya que para representar una salida se emplea el símbolo el cual tiene una función similar a la de una bobina en un relevador, la cual una vez energizada provoca un cambio de estado en él o en los interruptores que se encuentran bajo su influencia. Para programar un PLC, primero hay que contemplar las entradas y las salidas totales que estarán interactuando en el sistema que se va a automatizar, posteriormente es necesario plantear el procedimiento mediante el cual se relacionaran las entradas con las salidas de acuerdo con las respuestas que se esperan del sistema.

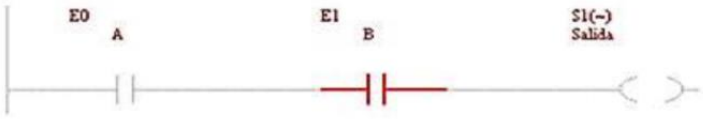
Una herramienta que se emplea frecuentemente para programar un PLC con las Tablas de Verdad, ya que en estas se observa la respuesta que debe emitir el PLC en función de las combinaciones de los estados lógicos de las entradas. La combinación generada por la forma en cómo se conectan las variables de entrada da origen a funciones lógicas estandarizadas como son: AND, OR, INVERSOR, etc. Las funciones lógicas mencionadas como todas las demás tienen asociado un símbolo por medio del cual se identifican en el área de la electrónica, las cuales son llamadas por su nombre en inglés y es la forma en nos referimos a ella.

Cuando se utiliza el Lenguaje en Escalera para programar un PLC no se emplean los símbolos de las funciones lógicas, por ello hay que implementarlas utilizando las variables de entrada y salida que de acuerdo con cierto arreglo se comportarán como las funciones lógicas: AND, OR, INERSOR, NOR, etc. Existen tres funciones lógicas a partir de las cuales se generan todas, las cuales se implementan, así como su comportamiento de la siguiente forma:

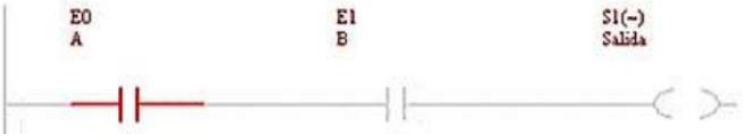
Función lógica "AND" (Y). Tendrá la salida activada (energizada) solo si ambos contactos (normalmente abiertos) tienen el nivel lógico de 1, en todos los otros casos la salida estará desactivada (desenergizada).



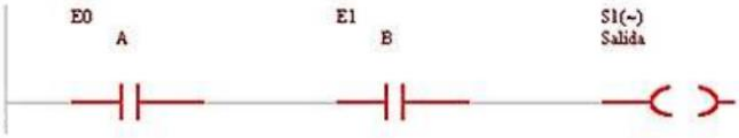
Función lógica AND (Y) con las entradas A y B en "0"



Función lógica AND (Y) con entrada A en "0" y B en "1"



Función lógica AND (Y) con entrada A en "1" y B en "0"



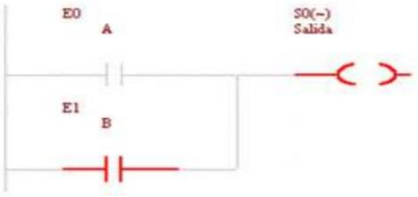
Función lógica AND (Y) con entrada A en "1" y B en "1".

Se genera la siguiente Tabla de Verdad:

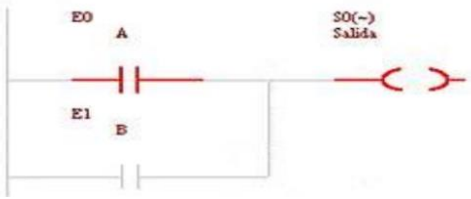
A	B	SALIDA
0	0	0
0	1	0
1	0	0
1	1	1

Función lógica OR (O). La salida se representa activada (energizada) si uno o todos sus contactos (normalmente abiertos) se encuentran en el estado de "1" lógico. En contraparte la salida se presentará desactivada (desenergizada) cuando todos los interruptores tienen un estado lógico "0".

Función lógica OR (O) con las entradas A y B en "0".



Función lógica OR (O) con la entrada A en "0" y B en "1"



Función lógica OR (O) con la entrada A en "1" y B en "0"

Función lógica OR (O) con las entradas A y B en "1"

Se genera la siguiente Tabla de Verdad:

A	B	SALIDA
0	0	0
0	1	1
1	0	1
1	1	1

Función lógica inversora (NOT). A diferencia de las funciones AND y OR, solo requiere un contacto en la entrada, el cual debe ser normalmente cerrado. La salida se presenta activada (energizada) si el contacto se encuentra en el estado de "0" lógico. En contraparte la salida se presentará activada (desenergizada) cuando el interruptor tiene un estado lógico "1". La finalidad de esta función lógica es presentar en la salida el estado lógico del contacto de manera invertida.

Función lógica inversora NOT con la entrada A en "0"

Función lógica inversora NOT con la entrada A en "1"

Se genera la siguiente Tabla de Verdad:

A	SALIDA
0	0
1	1

Función lógica no inversora. Requiere únicamente de un contacto, el cual debe ser normalmente abierto. La salida es el reflejo del estado lógico en el que se encuentre el contacto.

Función lógico NO inversora con la entrada A en "0"

Función lógica NO inversora con la entrada A en "1"

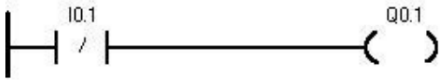
Variables binarias. Se representan mediante contactos, a cada uno de los cuales se asigna una identificación. Los contactos normalmente abiertos se representan de la siguiente forma: y representan variables directas.

Las variables inversas se representan mediante contactos normalmente cerrados de la siguiente forma:

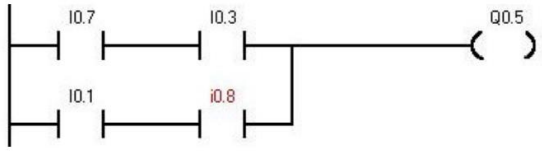
Secuencias lógicas. Las distintas funciones lógicas se pueden representar en este lenguaje, las cuales se mencionan a continuación:

- Función de una variable de entrada directa. Esta función se representa mediante un contacto normalmente abierto, que en forma general activa una variable de salida.

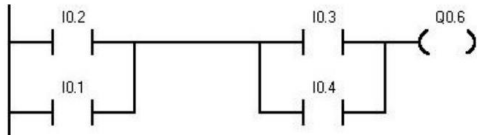
- Función de selección de una variable invertida. Esta función se representa mediante un contacto normalmente cerrado que activa una variable de salida.



- Función OR – AND. Esta función se representa mediante la combinación en paralelo de contactos conectado en serie.



- Función AND – OR. Esta función se representa mediante la conexión en serie de contactos conectado en paralelo.

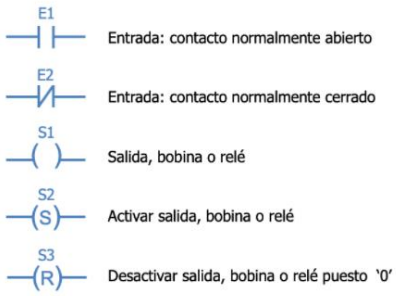


Para programar un autómata con Lenguaje Escalera, además de estar familiarizado con las reglas de los circuitos de conmutación, que es la lógica de contactos, es necesario conocer cada uno de los elementos de que consta este leguaje. A continuación, se describen de forma general los más comunes:

❖ Contacto normalmente abierto (E1). Si la variable asociada E1 vale "0", el contacto permanece abierto, si vale "1" se cierra.

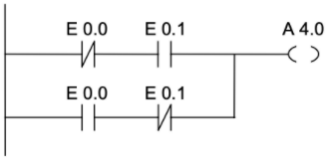
- Contacto normalmente cerrado (E2). Si la variable asociada E2 vale "1", el contacto permanece abierto y si vale "0" se cierra.
- Salida bobina o relé (S1). La variable asociada S1 tomará el valor de la variable o la combinación de variables, que esté a su entrada en el punto de conexión del lado izquierdo. También se puede enclavar o desenclavar indicándolo con una S o R.

Elementos básicos del Lenguaje Escalera:



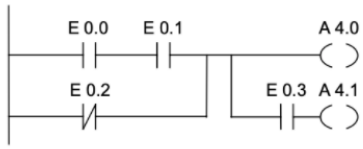
Ejemplos:

Circuito:



Equivale a: $A4.0 = \overline{E0.0} * E0.1 + E0.0 * \overline{E0.1} = E0.0 \otimes E0.1$ que corresponde con una puerta XOR.

Circuito:



Equivale a: $A4.0 = E0.0 * E0.1 + \overline{E0.2}$ y $A4.1 = E0.3 * (E0.0 * E0.1 + \overline{E0.2})$

Bibliografía.

https://instrumentaciónycontrol.net/funciones-logicas-y-and-y-o-or/

http://www.ieec.uned.es/investigacion/Dipseil/PAC/archivos/Informacion_de_referencia_ISE6_1_2.pdf

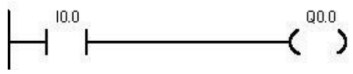
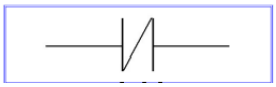
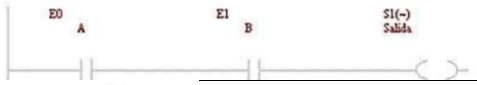
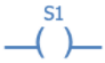
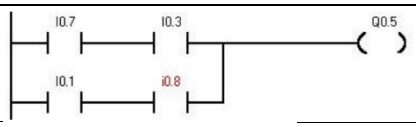
https://crustymks.com/es/industrial-automation/1058-plc-ladder-logic-functions-for-electrical-engineers-beginners.html

4.1 Diagrama de escalera | Introducción a la Automatización Industrial (bookdown.org)

Siemens, Esquema de Contactos (Kop) Para S7-300 Y S7-400, Siemens, 2004a.

Actividad 4. Lenguaje escalera popular en la programación de PLC

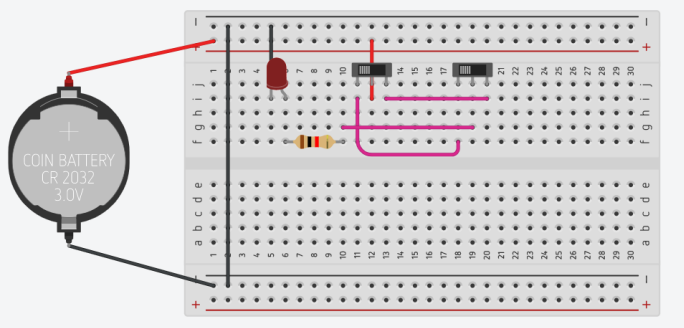
Instrucciones: Después de realizar la Lectura 4 relaciona las columnas con los elementos del Lenguaje Escalera.

Símbolo	Nombre
1. 	( ) Función lógica inversora NOT
2. 	( ) Función lógica AND
3. 	( ) Función lógica OR
4. 	( ) Entrada contacto normalmente abierto
5. 	( ) Entrada contacto normalmente cerrado
6. 	( ) Variable directa
7. 	( ) Función lógica no inversora
8. 	( ) Función OR – AND
9. 	( ) Variable inversa
10. 	( ) Salida, bobina o relé
11. 	( ) Función de una variable de entrada directa

Ejercicio 3. Circuito de escalera en Tinkercad

Instrucciones: Realiza el Ejercicio 3 siguiendo los pasos que se especifican.

Circuito:



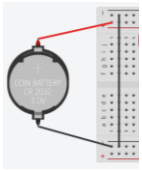
Componentes:

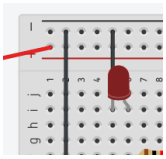
- Placa de pruebas pequeña.
- Pila plana de 3V.
- 1 led rojo.
- 1 resistencia de 1 Kohm
- 2 interruptores deslizantes.
- Cable negro, rojo, rosa.

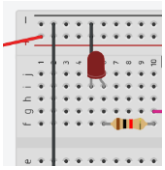
Procedimiento:

1. Conectar la pila plana a la placa con cable rojo a energía y negro a tierra, de la siguiente forma.

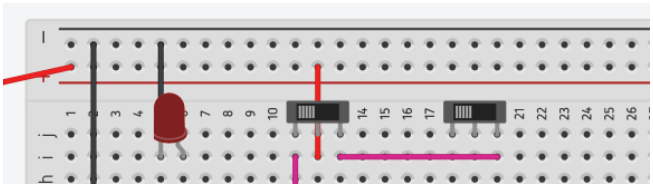
2. Realiza un puente de tierra a tierra de la siguiente forma.



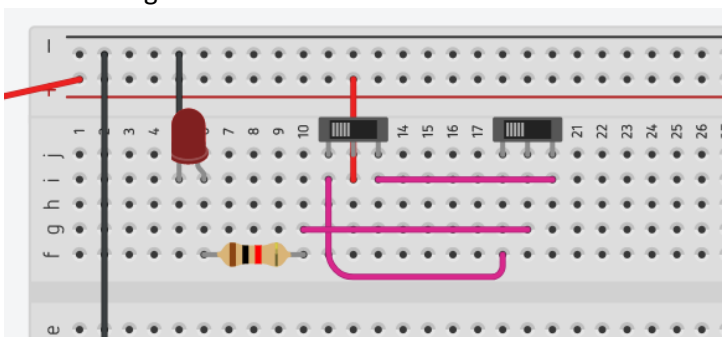
3. Conectar el led y conectar un cable a tierra como se muestra. 

4. Conectar la resistencia de la siguiente forma. 

5. Conectar los dos interruptores deslizantes en la placa y energía de la siguiente forma.



6. Realizar el siguiente cableado en escalera.

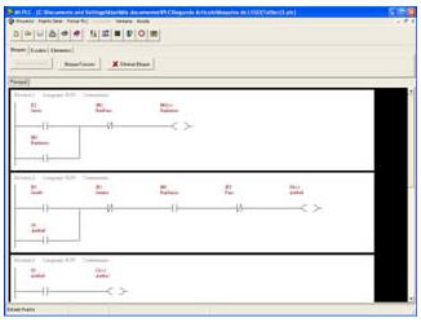


7. Ejecuta el circuito y llena la siguiente tabla de verdad.

A	B	SALIDA
0	0	
0	1	
1	0	
1	1	

8. De acuerdo con la tabla de verdad determina que compuerta lógica es y dibuja el diagrama en escalera.

Lectura 5. Programación escalera en el software MiPlc



Para programar un PLC se necesita conocer bajo que ambiente de programación se hará. Normalmente ese ambiente de programación es gráfico y se le conoce con el nombre de Lenguaje de Escalera, pero su nombre es Diagrama de Contactos. Existen diversos lenguajes de programación para los PLC, pero el Lenguaje Escalera es el más común y prácticamente todos los fabricantes de PLC lo incorporan como lenguaje básico de programación. El Lenguaje Escalera es el mismo para todos los modelos existentes de PLC, lo que cambia de fabricante a fabricante o de modelo a modelo es el microcontrolador que emplea, por ello lo que difiere entre los PLC es la forma en que el software interpreta los símbolos de los contactos en Lenguaje Escalera.

El software de programación es el encargado de generar el código en ensamblador del microcontrolador que posee el PLC. Para cada PLC el código que se crea es diferente, ya que por naturaleza propia los códigos de los microcontroladores son diferentes, aunque e Lenguaje Escalera sea el mismo para todos los PLC.

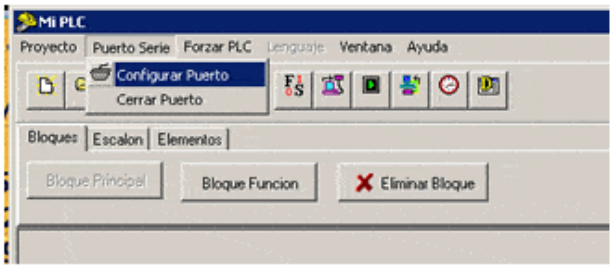
Se puede utilizar cualquier modelo de PLC, inclusive el fabricado por cualquier fabricante, dependiendo del PLC seleccionado, puede tener inclusive desde 6 entradas y 6 salidas. Debemos tomar en cuenta la cantidad de entradas y salidas que posea el PLC.

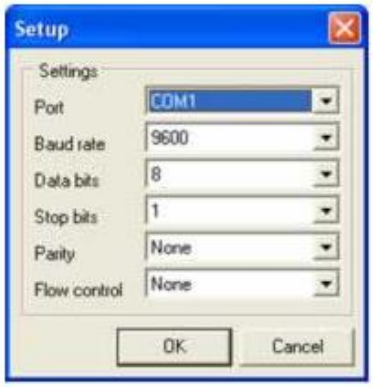
Cinda Software pone a disposición un software para PLC con simulador que tiene como característica importante, la de poseer la misma capacidad de trabajo que cualquiera de marca reconocida sea Allen Bradley o Siemens por ejemplo y está en español.

Para programar el PLC en este caso podemos simularlo con una aplicación industrial o con un programa de prueba. La primera acción para realizar es abrir el software de programación llamado "MiPlc", el cual debe estar instalado, se puede descargar del vínculo: http://www.instrumentoacionycontrol.net/Descargas/Cinda_MiPLC.zip o se solicita al docente.

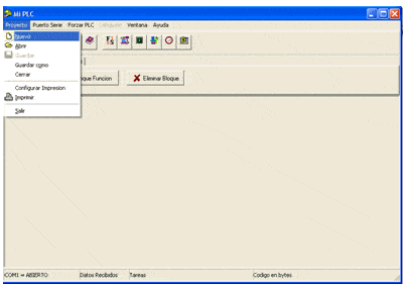
Al dar doble clic sobre el icono del software de programación MiPlc aparece una ventana de bienvenida en la cual se observan los datos de la empresa fabricante del PLC, con todos los datos necesarios de contacto, se da clic en el cuadro OK para ingresar al programa.

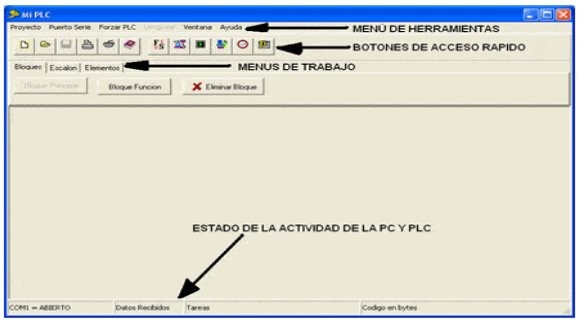
Al estar en el programa hay que dar clic en el menú de herramientas y seleccionar el que se llama Puerto Serie, como paso siguiente se tiene que seleccionar la opción de Configurar Puerto, con lo cual se abre la ventana etiquetada como setup, en la cual configuramos las características de la comunicación serial que se establecerá entre el PLC y la computadora por lo que normalmente se dejan los datos, cuando se tienen ingresados estos datos se da clic en el botón OK, con lo cual se abre el canal de comunicación serial. Con esto el software de nuestro PLC ha sido configurado adecuadamente para que pueda operar, lo que sigue es ingresar los símbolos correspondientes al programa.

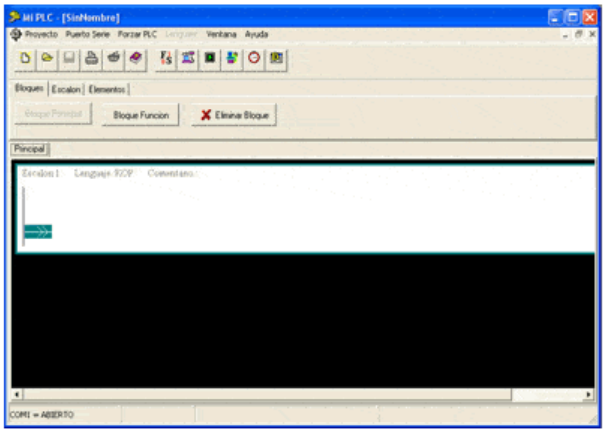




Se observa la imagen del software de programación de PLC en donde se identifican las partes que lo componen y son las siguientes: menú de herramientas, botones de acceso rápido, los menús específicos de trabajo y el estado de la actividad existente entre el PLC y la computadora.

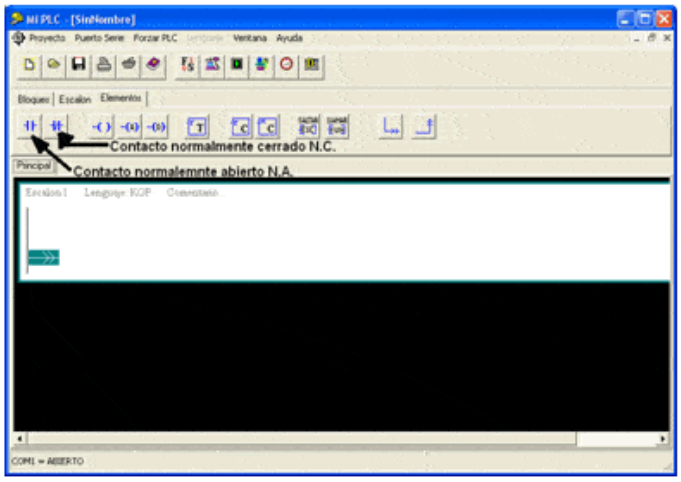




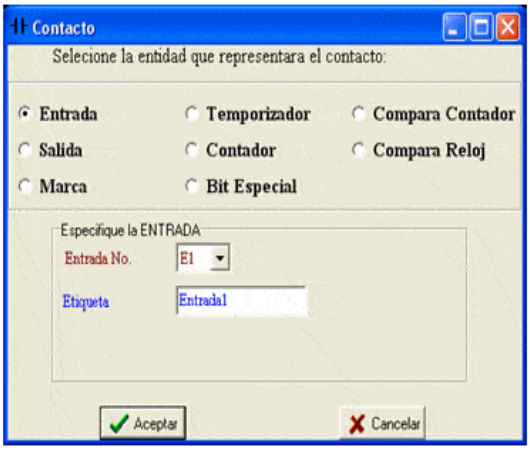


Para iniciar con un programa se tiene que crear un nuevo proyecto, por lo que hay que dar clic en el en menú Proyecto, posteriormente clic en el comando Nuevo. Cuando se abre un nuevo escalón estamos en posibilidad de comenzar a insertar los símbolos correspondientes al lenguaje en escalera para formar nuestro programa. Para ello se selecciona el menú específico de trabajo denominado "Elementos", porque en esta sección se tienen los símbolos que representan las operaciones que el programa tiene que ir interpretando, se describe símbolo por símbolo:

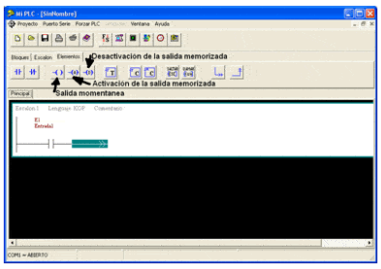
- El primer conjunto de símbolos corresponde a variables de señales de entrada, estas se denominan como contacto normalmente abierto (NA) y contacto normalmente cerrado (NC), su función principal es la de informar al PLC el estado lógico en que se encuentran las variables físicas que son captadas a través de sensores, al igual que los contactos de un relevador cuando este se encuentra desenergizado el contacto NA se encuentra abierto, mientras que el contacto NC se encuentra cerrado y cuando se activan el contacto NA se cierra y el contacto NC se abre, existe un cambio de estado cuando los contactos son manipulados.



Estos contactos constituyen las "Condiciones" que sirven para generar la lógica de programación del PLC, por medio de estos se implementan las funciones lógicas que el programa de control de algún proceso industrial utiliza. Para insertar alguno de estos símbolos basta con seleccionarlo con el puntero del ratón y dar clic con el botón izquierdo, esta acción provocará que se abra una ventana preguntando qué tipo de entrada es, por ello se selecciona si se trata de una entrada a través de los bornes de conexión (entrada física) o se trata de una entrada interna (estado generado por alguna operación interna del PLC). Cuando se selecciona el tipo de entrada tendremos que indicar de donde leerá la información por lo que tenemos que seleccionar el origen de la entrada (sea física o interna) y por último asignarle una etiqueta que corresponda a la información que está leyendo.



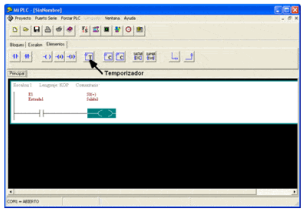
- El segundo conjunto de símbolos corresponde a variables de salida, las que a su vez activarán elementos de potencia, mismos que pueden ser motores de CD o de CA, calefactores, pistones, lámparas, etc. Los símbolos que se emplean para representar a las salidas en el lenguaje en escalera tienen el mismo significado que en un diagrama eléctrico tiene la bobina de un relevador, lo mismo que sucede con uno real para que se energice se tienen que cumplir ciertas condiciones lógicas previas, así sea accionar un botón. Los símbolos que se activan a las salidas constituyen las "Acciones" que todo proceso industrial debe



efectuar, esto es para modificar las variables físicas que se encuentran interviniendo en cualquier línea de producción. Las salidas dependiendo de cómo se lleve a cabo su manejo de memoria, reciben los nombres de salida momentánea o salida memorizada.

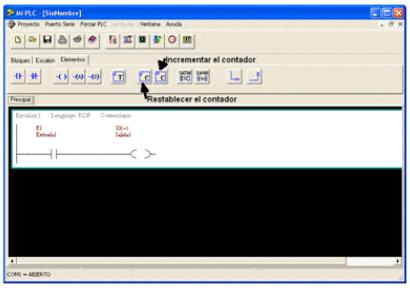
La salida momentánea nos representa un estado lógico que hará encender o apagar cualquier elemento actuador, esta salida se caracteriza por el modo de operación que nos dice que para tener un "1" lógico a la salida es requisito indispensable el que las Condiciones que prevalecen a la entrada se mantengan todo el tiempo que sea necesario para que ese "1" lógico exista, de cualquier otra forma lo que se tendrá es un "0" lógico a la salida. La salida memorizada contiene de manera implícita una memoria, la cual es de mucha utilidad para mantener para mantener el estado de "1" lógico durante todo el periodo de tiempo que el proceso así lo requiera, lo único que se debe hacer es activar la salida con memoria, cuando se activa la salida memorizada no importa que cambien las Condiciones el estado de "1" lógico no se modifica. Ahora, bien, cuando sea necesario que se tenga que cancelar la memoria o también se puede expresar que se apagará la salida o se llevará al estado de "0" lógico, lo que se tiene que realizar es accionar la desactivación correspondiente.

Cuando se utiliza una salida se tienen dos posibilidades de configurarla, un tipo de salida es como externa por lo que se define como salida, para ello se indica a que terminal física del bornero de conexión está reflejándose su actividad. El segundo tipo de salida es considerada como interna y se denomina marca, lo que representa es que esta marca es una condición interna del programa de control que no tiene reflejo hacia algún elemento actuador. En algunos PLC sólo se permite tener un símbolo de salida, si se requiere más de uno, se necesita abrir tantos escalones como salidas tengamos en nuestro proceso.

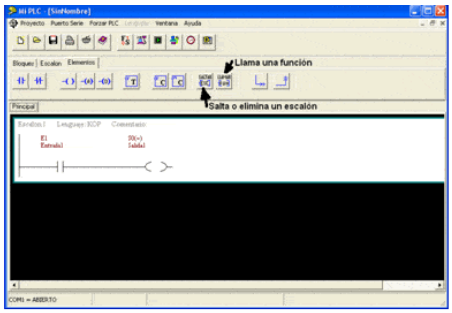


- El tercer conjunto de símbolos está compuesto por uno sólo y se trata del temporizador, el cual es una herramienta que tiene la función de activar el cinteo de un intervalo de tiempo que tiene como base 1 segundo, el tiempo máximo que se puede fijar es el de 255 segundos. El temporizador es una gran ayuda sobre todo cuando se pretende establecer una condición de seguridad para el operador, por ejemplo, cuando transcurre un tiempo de algunos segundos sin que exista respuesta alguna, entonces el accionamiento de los botones de control no responderá si no hasta que el proceso se restablezca. El temporizador una vez que es activado comienza a cuantificar el tiempo de forma descendente y cuando llegue a 0 segundos origina una salida interna con el estado de 1 lógico, cancelándose esta salida cuando se restablece el temporizador.
- El cuarto conjunto de símbolos sirve para utilizar la herramienta que tiene la función de contar eventos, a este contador se le tiene que fijar cual es el valor máximo al que tiene que llegar que,

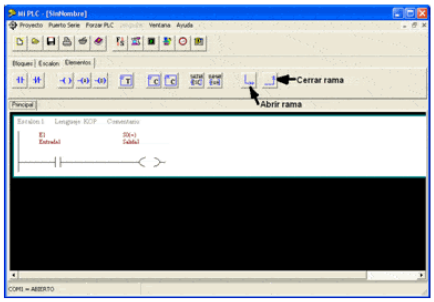
dependiendo del PLC, pero normalmente para controlar el proceso de llenado de una caja de productos no se requieren de valores muy altos. Una vez que fue activado y llega a su conteo máximo se origina una salida interna con el estado de 1 lógico avisando que ha llegado al valor de conteo prefijado para colocar en 0 lógico la salida interna del contador, este se debe de restablecer para poder comenzar con un nuevo proceso de conteo.



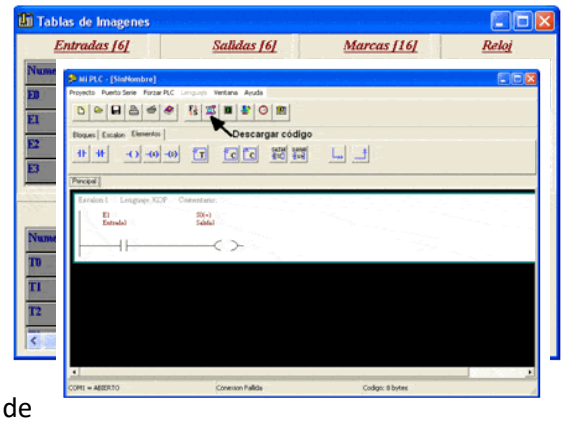
- El quinto conjunto de símbolos está integrado por dos herramientas, una que sirve para diseñar funciones que operen a manera de subrutinas y otra que sirve para saltar un escalón, que es lo mismo que inhabilitarlo. Las subrutinas se emplean cuando en el desarrollo de nuestra aplicación, existen condiciones que se repiten más de una vez, si las ingresamos en cada escalón diferente nos llevaría a incrementar enormemente nuestro programa, razón por la cual para simplificarlo se diseña una función que internamente contenga toda la lógica de control que se repite constante y posteriormente solo se llama y no se ingresan todos los símbolos. La segunda herramienta que sirve para saltar un escalón se emplea cuando dependiendo del contexto del programa de control lógico, cuando una condición se lleva a cabo conlleva el seleccionar uno de dos o más caminos, por lo que se selecciona el adecuado y se eliminan los demás.



- El sexto y último conjunto de símbolos sirve para realizar bifurcaciones cuando se están ingresando los contactos ya sean NA o NC. Estos símbolos sirven para abrir una rama y también para cerrarla.



Cuando se ingresan todos los símbolos de nuestro programa en lenguaje en escalera, es recomendable antes de programar al PLC simular las funciones lógicas y tener la certeza de que nuestra lógica funciona por lo que hacemos uso de la tecla de acceso rápido correspondiente, como respuesta de la sección anterior se provocará que una ventana se abra visualizando ahí el estado que guardan todas las entradas, salidas, temporizadores, contadores, etc.



Para realizar la simulación de nuestro programa tenemos que ir manipulando en el recuadro correspondiente las condiciones, o sea las entradas y tan solo basta con que coloquemos el apuntador del ratón y dar clic con el botón izquierdo del mismo para cambiar el estado lógico que contenía. Cuando se ha simulado el programa y este ejecuta todas las condiciones lógicas que le programamos, ya estamos en posibilidad de cargar el programa al PLC, por lo que ahora conectamos el cable de programación tanto al puerto serie de la computadora como a la terminal correspondiente del PLC, para ello hacemos uso del botón de acceso rápido.

Bibliografía.

Aprende PLCs desde Cero a Avanzado con RsLogix500 | MyAutomationClass.

https://instrumentacionycontrol.net/

http://www.ieec.uned.es/investigacion/Dipseil/PAC/archivos/Informacion_de_referencia_ISE6_1_2.pdf

Siemens, Esquema de Contactos (Kop) Para S7-300 Y S7-400, Siemens, 2004a.

Actividad 5. Programando mi PLC en lenguaje escalera

Instrucciones: Después de realizar la Lectura 5 implementa la función lógica AND LOD en Lenguaje Escalera.

Compuerta lógica de una función AND LOD.

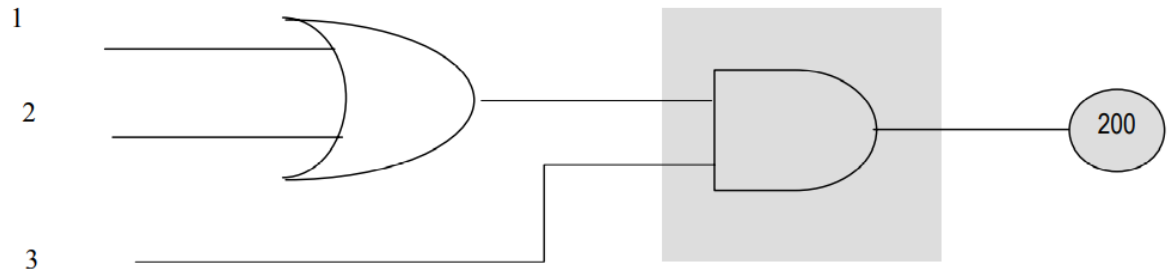


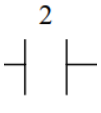
Tabla de verdad de una función AND LOD.

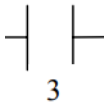
Realiza la combinación de las salidas 1 y 2 para completar la tabla de verdad.

E1	E2	E3	S1	S2
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	1	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

Componentes para representar en el diagrama en lenguaje escalera.

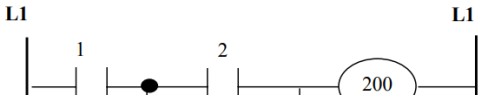
- LOD 1 

- LOD 2 

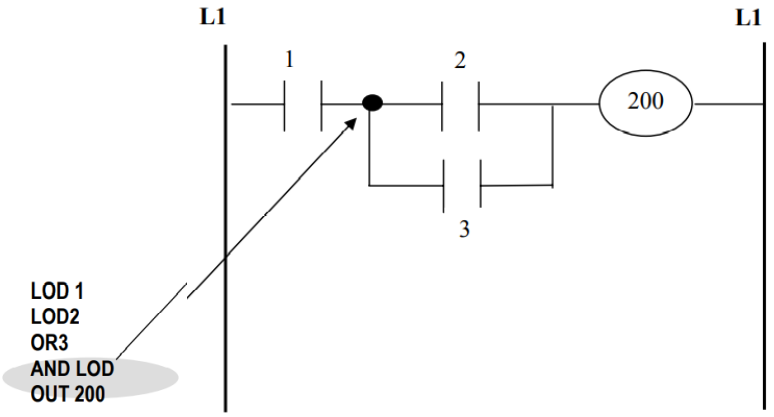
- OR3 

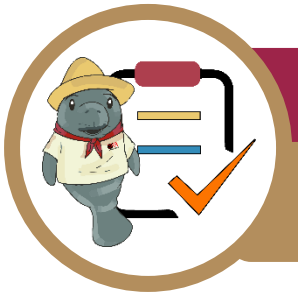
- AND LOD 

- OUT 200 

- Escalón 1 

Realiza el diagrama en lenguaje escalera utilizando los símbolos especificados. Después de realizarlo capturarlo en el software MiPlc y ejecutar la simulación.





INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 5

Actividad 5: Programando mi PLC en lenguaje esclera

DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Diagrama		**Fecha**	
Formación laboral básica: Robótica **Submódulo 2:** Control industrial		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente**	

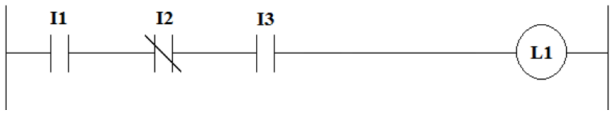
No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Realiza las combinaciones de la tabla de verdad.			2		
2.	Realiza el diagrama en lenguaje escalera con los símbolos especificados.			2		
3.	Realiza el diagrama en lenguaje escalera en el software MiPlc.			2		
4.	Coloca la función lógica AND LOD de la forma correcta.			2		
5	Ejecuta el circuito y funciona en forma correcta.			2		
				CALIFICACIÓN		

Ejercicio 4. PLC en lenguaje escalera

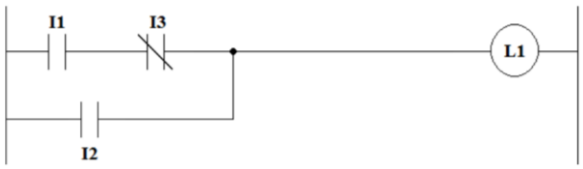
Instrucciones: Realiza el Ejercicio 4 resolviendo cada uno de los problemas que se especifican en el software MiPlc con la simulación correspondiente.

Ejemplo:

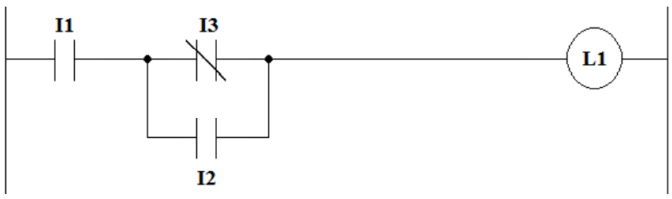
Con la expresión lógica para la salida L1: L1 = (I1) and (not I2) and (I3), realiza el diagrama en lenguaje escalera.



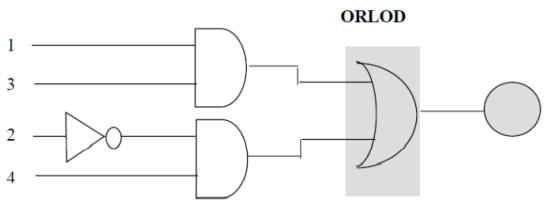
1. Con la expresión lógica para la salida L1: L1 = ((I1) and (not I3)) or (I2).



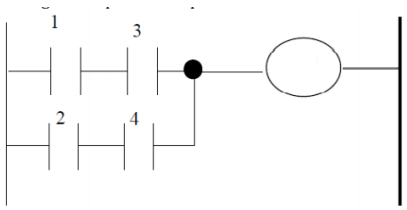
2. Con la expresión lógica para la salida L1: L1 = (I1) and ((not I3) or (I2)).



3. Con el circuito lógico:

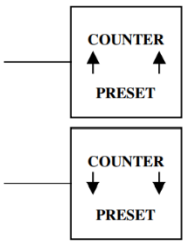


realiza el diagrama escalera.



Lectura 6. Contadores y temporizadores en PLC

Contadores.



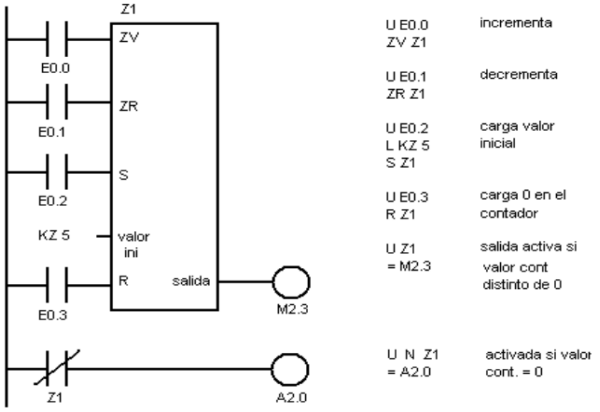
Son posiciones de memoria que almacenan un valor numérico, mismo que se incrementa o decrementa según la configuración dada a dicho contador. Un contador debe tener un valor prefijado como meta o Preset, el cual es el número que el usuario programa para que dicho contador sea activo o inactivo según el valor alcanzado.

Para la realización de tareas de conteo de eventos externos los PLC tienen los contadores. Estos los hay de diferente tipo, al igual que los que se fabrican en circuitos TTL, hay contadores incrementales, decrementales, conteo UP DOWN, etc. Los diferentes PLC nos proporcionan algunos o a veces todos estos tipos de contadores.

Por ejemplo, si el contador tiene un Preset de 15 y el valor del conteo va en 14 se menciona que el contador se encuentra inactivo, con ello no se dice que no esté contando. Pero al siguiente pulso, cuando el valor llegue a 15, el contador es activo porque ha llegado al valor de Preset. Dependiendo del software, puede ocurrir que el contador empiece en su valor de Preset y cuente hacia abajo hasta llegar a cero, momento en el cual estaría activo. Nos permitirán contar y/o descontar impulsos que enviemos al contacto que lo activa entre 0 y 999. Los parámetros son:

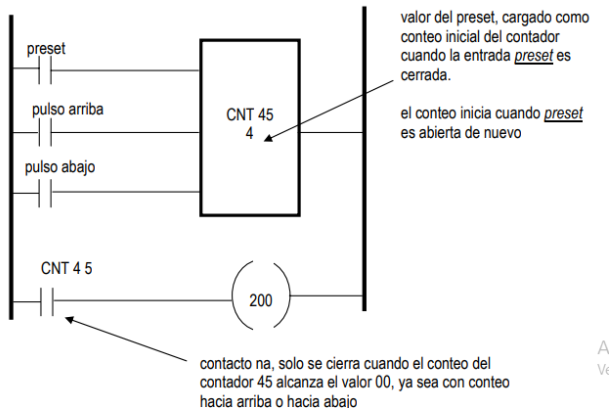
- Z0…MAX. Número de contador.
- ZV. Incrementa el valor del contador no superando el valor de 999.
- ZR decrementa el valor del contador no decrementa por debajo de 0.
- S. Carga el valor inicial del contador.
- KZ xxx. Valor inicial.
- R. Resetea el valor del contador.

La salida del contador estará a "1" siempre que el valor del contador sea diferente de "0".

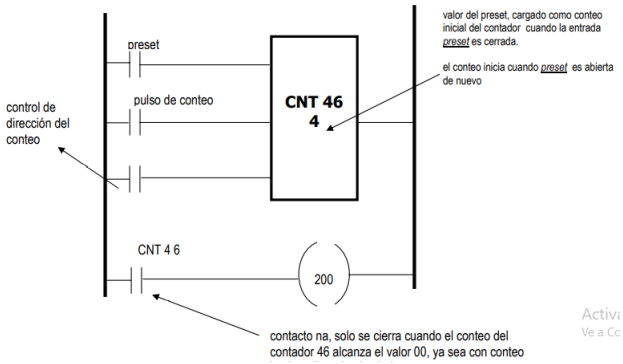


Se muestran algunos contadores usados por el MICRO1 de SquareD:

Contador reversible (Up-Down)
Contador 45



Contador reversible (Up-Down)
Contador 46



Las opciones de programación de los contadores son:

- Asignación 

. Con este elemento se define el nombre del contador a ser utilizado y el valor inicial de la cuenta.

- Cuenta Ascendente 

. Un flanco de subida en la entrada del elemento hace que el valor de la cuenta se incremente en uno. El flanco de subida se define como el cambio de una señal de F a V.

- Cuenta descendente 

. Con un flanco de subida se hace que el valor de la cuenta descienda en uno.

- Reposición 

. Obliga a que el contador se reinicie con su valor inicial.

La salida de un contador es un contacto cuya variable de referencia sea el nombre del contador, la variable es F mientras el valor de la cuenta sea 0 y es V si la cuenta es diferente de 0.

Los dispositivos contadores se utilizan para realizar operaciones de cuenta, de tipo ascendente, descendente o incluso ascendente/descendente. El valor de la cuenta se actualiza cada vez que un flanco de subida se aplica en una determinada entrada. La salida del dispositivo se sitúa en estado lógico alto cuando se alcanza el valor de cuenta predefinido. Varían en función de marcas y modelos, pero los más usados suelen incorporar 32 contadores: Z0 … Z31.

De los 32 contadores, 8 no se borran al desconectar el autómata son remanentes, dichos contadores son Z0 a Z7. Para consultar el estado de cada uno de ellos se pueden usar como si fueran entradas mediante operaciones combinacionales o introduciendo su valor en los AKKU.

Temporizadores.

Un temporizador es un dispositivo electrónico utilizado para proveer señales de base de tiempo o para generar señales de acción retardada variable. Un temporizador o Timer digital consiste generalmente de un contador decreciente en donde cada decremento en uno del Preset del contador, será realizado a una frecuencia conocida (veces por segundo) y al llegar a cero se activa un relevador interno o uno de salida.

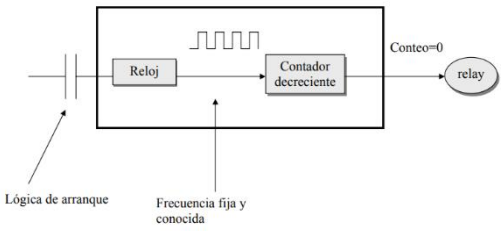
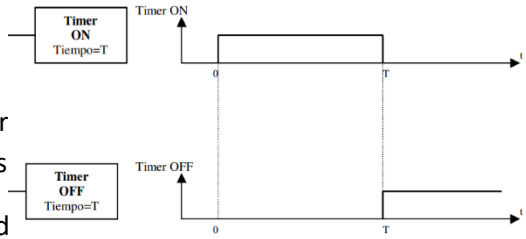


Diagrama de bloques de las componentes de un timer conceptual

A como indica su nombre, cada vez que alcanzan cierto valor de tiempo activan un contacto interno. Dicho valor de tiempo, denominado Preset o meta, debe ser declarado por el usuario. Cuando se indica el tiempo de meta, se debe indicar con cuales condiciones debe empezar a temporizar, es decir a contar el tiempo. Para ello, los temporizadores tienen una entrada denominada Start o inicio a la cual deben llegar los contactos o entradas que sirven como condición de arranque. Dichas condiciones, igual que cualquier otro renglón del diagrama escalera, pueden contener varios contactos en serie, en paralelo, normalmente abiertos o normalmente cerrados. Una de las tantas formas de representación sería:



Las operaciones de tiempo per rnos del autómata. Existen diversos tipos de temporizadores de parámetros:

- Arranque del temporizad mporizador, contactos como se desee.

- Carga del tiempo. La forma habitual es mediante una constante de tiempo, pero puede haber otros ajustes, por ejemplo: leyendo las entradas, un valor de una base de datos, etc. Esta carga de valor se debe realizar con la instrucción L que lo almacena en una zona de memoria llamada acumulador (AKKU1) para después transferirlo al temporizador.

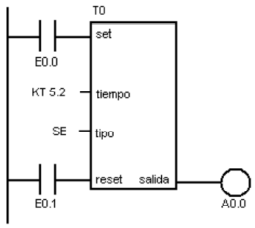
Formato L KT xxx.yy KT constante de tiempo. Xxx a tiempo (máximo 999) y a base de tiempos 0 = 0.01 segundos (centésimas). 1 = 0.1 segundo (décimas). 2 = 1 segundo. 3 = 10 segundos. En donde los segundos se multiplican por 10 (segundos * 10).

Ejemplo: KT 243.1 24, 3 segundos. KT 250.2 250 segundos.

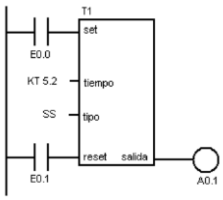
- TO…MAX. Número de temporizador, el número MAX depende del fabricante.
- Paro del temporizador. Es opcional y pone a cero el valor contado en el temporizador.

Se muestran diferentes tipos de temporizadores:

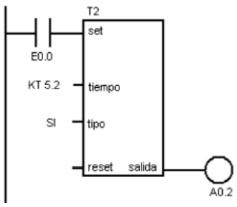
- SE. Con retardo a la conexión. Manteniendo la entrada set a 1. La entrada reset desconecta el temporizador.



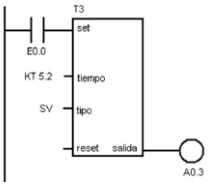
- SS. Con retardo a la conexión activado por impulso en set. Sólo se desconectará la salida por la entrada reset.



- SI. Mientras mantenemos conectada la señal set, la salida estará activada durante KT.



- SV. Mantiene la salida activa durante KT independientemente del tiempo que la señal set esté activa.



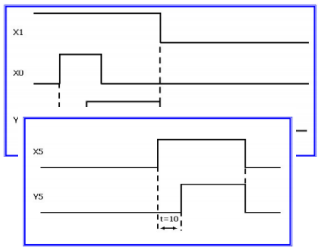
TMR, realiza la función de temporización, utiliza para ello dos variables:

- Xi = Variable de puesta a cero.
- Xj = Variable temporizada.

La salida del temporizador se realiza a través de una variable de salida externa o interna. La programación del temporizador necesita cuatro instrucciones en secuencia, una instrucción de selección de la variable de puesta, una instrucción de la variable temporizada, la instrucción TMRn que elige el temporizador. TMRn inicia la temporización si Xi está en 1 y Xj pasa a ser 1, es decir se activa la variable cuyo cambio se temporiza, una disposición de memoria de programa que almacene el valor del tiempo preseleccionado.

Ejemplo: Diagrama de secuencia temporal de un temporizador.

Diagrama de secuencia temporal de un retardo a la conexión.



El tiempo de retardo (T#xx) se establece: En la parte superior del símbolo de disparo del temporizador en segundos o en milisegundos. Mediante el formato T#multiplicador.escala como producto entre la base de tiempo estipulada por la escala y el multiplicador. De esta forma se tiene que: Retardo=Base de tiempo*multiplicador

Observamos en la tabla los posibles valores de base de tiempo:

Valores Base de Tiempo		
Valor de Escala	Base de Tiempo	Ejemplo
0	0.01 S	T#20.0 Retardo = 0.2 S
1	0.1 S	T#15.1 Retardo = 1.5 S
2	1 S	T#30.1 Retardo = 30 S
3	10 S	T#60.3 Retardo = 600 S

La salida del temporizador es cualquier contacto al cual se le haya asignado como variable de referencia el nombre del temporizador.

Bibliografía.

Aprende PLCs desde Cero a Avanzado con RsLogix500 | MyAutomationClass.

https://instrumentacionycontrol.net/

http://www.ieec.uned.es/investigacion/Dipseil/PAC/archivos/Informacion_de_referencia_ISE6_1_2.pdf

Siemens, Esquema de Contactos (Kop) Para S7-300 Y S7-400, Siemens, 2004a.

Actividad 6. Contadores y temporizadores en Mi PLC

Instrucciones: Después de realizar la Lectura 6 resuelve el rompecabezas y escribe que sucede cuando presionas Star.

Para resolver el rompecabezas ingresa al siguiente vínculo:

https://puzzel.org/es/jigsaw/play?p=-MftabsIVmwllJtYVAdK

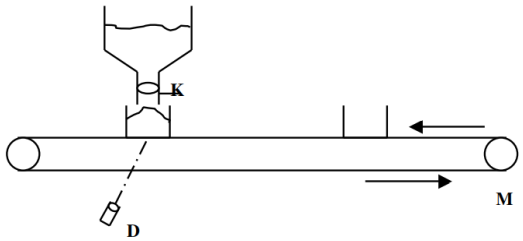
Escribe lo que ocurre cuando ejecutas este diagrama escalera.

Ejercicio 5. Temporizadores en lenguaje escalera

Instrucciones: Realiza el Ejercicio 5 resolviendo la problemática plateada y en el software MiPlc con la simulación correspondiente.

Problema planteado.

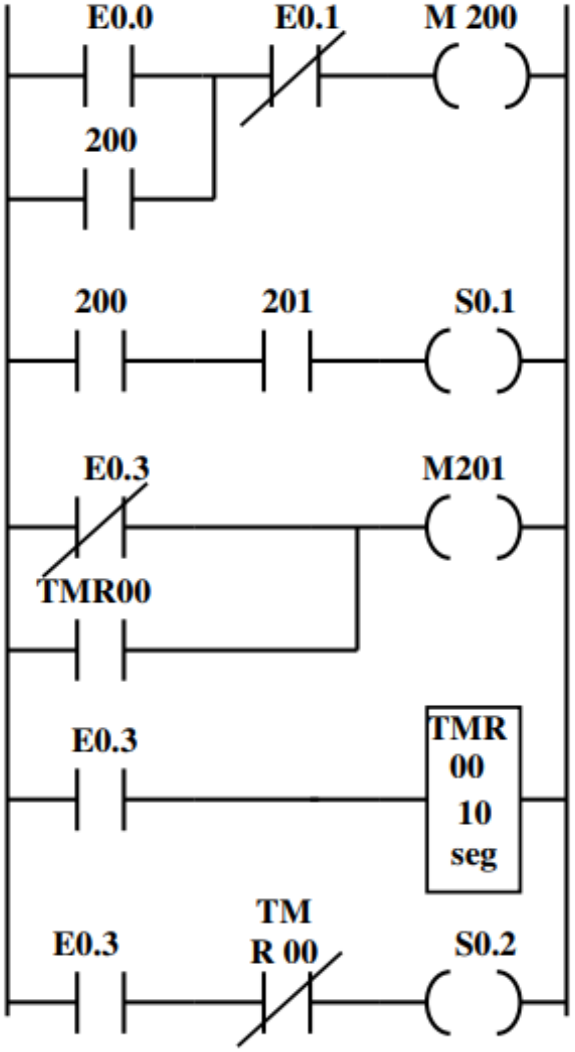
Sobre una cinta transportadora impulsada por un motor M, se transportan cajas las cuales deberán detenerse bajo una tolva al ser detectadas por un sensor D. Una vez detenida la caja bajo la tolva, se abrirá una esclusa (Mediante el contactor K1) durante 10 segundos., tiempo en el cual la caja se llena. Pasado este tiempo, la esclusa deberá cerrarse y la cinta comenzará a moverse quitando la caja de esa posición. Este proceso se deberá repetir cuando pase otra caja bajo la tolva. Se pide realizar el diagrama escalera y el cuadro de asignaciones. Nota: La esclusa se abre cuando es activado el contactor K1 y se cierra al desactivarse este. La cinta está funcionando siempre, salvo cuando una caja es detectada.



Cuadro de asignaciones.

E0.0	**Start**
E0.1	Stop
E0.3	Detector de la caja
S0.1	Motor de la cinta
S0.2	Contacto de la tolva (K1)
M200	Marca interna
M201	Marca interna
TMR00	Temporizador OFF de 10 segundos

Diagrama escalera.


E0.0
E0.1
M 200
200
200
201
S0.1
E0.3
M201
TMR00
E0.3
TMR
00
10
seg
E0.3
TM
R 00
S0.2

Ejercicio 6. Contadores en lenguaje escalera

Instrucciones: Realiza el Ejercicio 6 resolviendo la problemática planteada y en el software MiPlc con la simulación correspondiente.

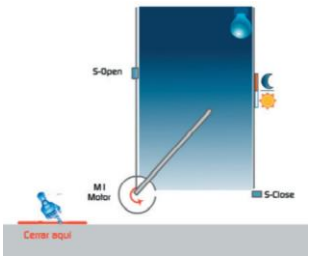
Problema planteado.

Control de apertura y cierre de puerta con luz de pasillo temporizada.

Para proveer el control automático de una puerta disponemos de los siguientes componentes:

- Motor M1 eléctrico para abrir y cerrar la puerta.
- Contactores.
  1) M ON, encendido y apagado el motor.
  2) M_open, direccionamiento del giro al motor necesario para abrir la puerta.
  3) M_close, direccionamiento de giro de cerrar la puerta.
  4) Interruptor ABRIR que accionado ordena la apertura de la puerta y sin accionar ordena que se cierre.
- Sensores.
  1) S_open, fin de carrera puerta totalmente abierta
  2) S_close, fin de carrera puerta cerrada.
  3) S_Día, tipo Día/Noche que se activa cuando hay suficiente luz solar. 3. Bombilla LUZ, para iluminar el pasillo.
  4) Interruptor ILUM, para el encendido manual de la luz del pasillo.
  5) Cuando se activa ABRIR la puerta debe abrirse y al desactivarse la puerta debe cerrarse.

Motor M1 eléctrico para abrir y cerrar la puerta.



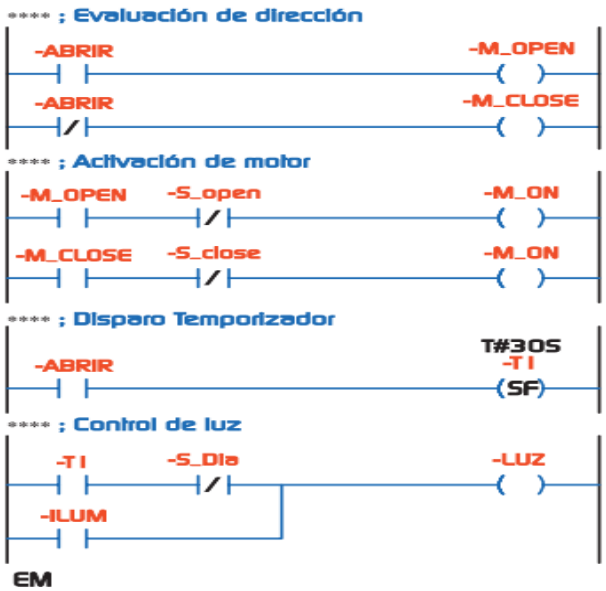
Si es de noche, la Bombilla LUZ debe encenderse durante 30 segundos, adicionales al comando de cierre de la puerta; ella también puede encenderse manualmente en cualquier momento.

Cuadro de asignaciones.

Símbolo	Variable	Descripción
M_ON	Q0.0	V = motor encendido F = motor apagado
M_open	Q0.1	Sentido de giro para abrir la puerta
M_close	Q0.2	Sentido de giro para cerrar la puerta
Abrir	I0.0	Interruptor V = orden de abrir F = orden de cerrar
S_close	I0.1	Sensor fin de carrera de puerta cerrada
S_open	I0.2	Sensor fin de carrera puerta abierta
S_Dia	I0.3	Sensor de luz solar
LUZ	Q0.3	Bombilla del pasillo
ILUM	I0.4	Interruptor de luz del pasillo
	T1	Temporizador de Retardo de desconexión

El estado de ABRIR dará los valores para M_open y M_close. El contactor del motor M_ON debe estar activo hasta alcanzar el fin de carrera S_close si M_close está activo, o hasta alcanzar S_open si M_open está activo. La LUZ debe encender si ABRIR está activo y S_Día es falso (noche) o si ILUM es activo. Además, ABRIR debe disparar un temporizador T1 de retardo a la desconexión por 30 segundos para mantener LUZ encendido.

Diagrama escalera.



Lectura 7. Simuladores de aplicaciones industriales

La construcción de simuladores inicia de manera oficial desde la época del renacimiento, donde se plantearon y resolvieron los primeros sistemas de simulación, relacionados básicamente con los juegos de azar y comprobación de resultados probabilísticos. Sin embargo, el uso actual de la palabra simulación data del año 1940, cuando los científicos Von Neuman y Ulam trabajaban en el proyecto Monte Carlo (basado en la obtención de datos de la ruleta rusa en Mónaco).

Durante la segunda guerra mundial, resolvieron problemas relacionados a las reacciones nucleares en las experimentaciones de la bomba atómica, cuya solución experimental sería muy costosa y cuyo análisis matemático sería demasiado complejo. Con la utilización de las supercomputadoras en el desarrollo de los experimentos de simulación, fueron surgiendo novedosas aplicaciones y como consecuencia de ello, una mayor cantidad de problemas teóricos y prácticos.

Los lenguajes utilizados para resolverlos no eran lo suficientemente aceptables, ya que dependían en gran cantidad a la experiencia científica de quienes lo creaban y su generalización era imposible. Se utilizaba lenguajes de máquina o ensamblador, con la dificultad de ser exactos en cuanto a fechas; posteriormente el avance de diseño de hardware y la aparición de lenguajes de propósito general y especiales, permitieron de algún modo generalizar en un porcentaje aceptable el uso y aplicación de los conceptos, métodos y técnicas de simulación.

La difusión del término de simulación y de sus aplicaciones se ha extendido en un vasto número de usuarios que siguen generando nuevos retos y proyectos para esta ciencia. Actualmente la interrelación de la simulación con otras técnicas, han permitido desarrollar proyectos más completos que vislumbran formidables perspectivas de transformación tecnológica.

en el área industrial se debe destacar que los primeros modelos los aplicaron en las fábricas para remplazar la mano de obra en actividades tales como anotaciones contables, escritura de informes y resolución de ecuaciones, obteniendo como ventaja la velocidad y la exactitud de sus cálculos.

A continuación, se mencionan varias definiciones de simulación tomada de diversos autores:

- "Simulación es el proceso de diseñar un modelo de un sistema real".
- "La simulación es un eficaz instrumento para el análisis y diseños de sistemas; permite la construcción de modelos que son una representación exacta del mundo real".

De acuerdo con estas definiciones se puede decir que la simulación no es más que la "herramienta que permite analizar, diseñar y evaluar un sistema a través de modelos que contemplen cualitativa y cuantitativamente las entradas y salidas del sistema".

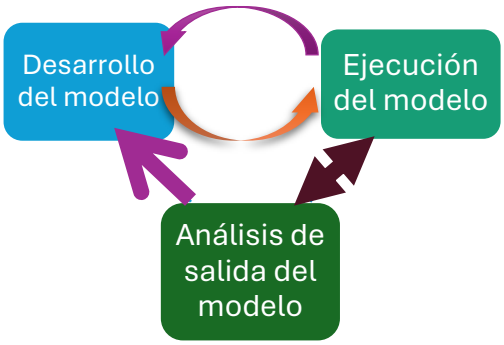
La simulación cada vez se vuelve un instrumento necesario en cualquier área de trabajo, con importancia en los siguientes:

- Es menos caro y más rápido que construir físicamente el sistema real.
- Descubrir errores de diseño en el modelo en lugar de hacerlo en el sistema real.
- Instrumento de estimación y pronóstico.
- Con base a resultados obtenidos de la simulación podemos tomar decisiones a tiempo.
- Estrategia de planeación.
- La simulación proporciona un control sobre el tiempo, debido a que es un fenómeno que se puede acelerar o retardar según se desee.

Desde cualquier punto de vista, el incluir métodos y técnicas de simulación en un proceso, sistema, procedimiento, etc., asegura un análisis mucho más conveniente tanto en consumo de recursos físicos como de logística; además de que se consiguen resultados confiables con un margen de error mínimo y evitando pérdidas producto de una planificación sin bases de conocimiento.

En la industria, la simulación se aplica en varias etapas, por ejemplo: en la etapa de diseño para ayudar con el mejoramiento de un proceso o diseño, o a su vez a un sistema ya existente para explorar algunas modificaciones. Es recomendable la aplicación de la simulación a sistemas ya existentes cuando hay algún problema de operación o bien cuando se requiere llevar a cabo una mejora en el comportamiento. El efecto que sobre el sistema ocurre cuando se cambia alguno de sus componentes se puede examinar antes de que ocurra el cambio físico en la planta para asegurar que el problema de operación se soluciona o bien para determinar el medio más económico para lograr la mejora deseada.

En la imagen, se visualiza las fases o etapas de simulación, donde se resaltan las interacciones entre los procesos principales: Desarrollo del Modelo, Ejecución del Modelo y Análisis de salida del modelo.



- El desarrollo del modelo permite formular el problema, definir el sistema simulador, formular modelos de simulación y trasladar al modelo a computadora.
- La ejecución de los modelos contempla la Validación, Verificación y Experimentación.
- Análisis de Salida del Modelo. Este paso permite la Implementación y Documentación de la simulación.

Uno de los propósitos de los modelos de simulación de procesos es la de identificar posibles mejoras en los procesos. El objetivo del análisis de proceso es primeramente el mejorar la comprensión (entendimiento) de los procesos, especialmente su estructura y su comportamiento dinámico. Un segundo objetivo es el de identificar opciones de mejoras de los procesos. Para ambos, la simulación sirve como un método apropiado porque nos permite experimentar con modelos de procesos en vez de procesos del mundo real realizando de diferentes formas. Se centran sobre los requisitos de simulación y parámetros necesarios para chequear las condiciones de consistencia de los interfaces de procesos y analizar los aspectos de comunicación de un software de un entorno de procesos dado.

Son muchas las aplicaciones de la simulación industrial y aunque para la mayoría de las personas esto les resulta totalmente transparente lo cierto es que allá donde miremos encontraremos algo relacionado con la simulación industrial.

Beneficios de la simulación.

- Permite realizar alteraciones en el modelo de simulación para observar y estudiar los cambios y efectos internos y externos del comportamiento del sistema
- Generalmente es más barato mejorar el sistema vía simulación, que hacerlo directamente en el sistema real.
- Debido a su bajo costo nos permite financiar proyectos costosos.
- Permite estudiar el sistema sin modificarlo, a través de la observación detallada de la simulación consiguiendo estrategias que mejoren la operación y eficiencia del sistema.
- Genera una visión macro y micro del sistema de forma general y detallada.
- Permitir tratar problemas planteados en amplios periodos de tiempo, comprimiendo su estudio a unos minutos.
- Carácter descriptivo, permite la realización de análisis de sensibilidad.
- Se puede utilizar en todos los niveles de la organización como: operativos, tácticos y estratégicos.
- La simulación por computadora permite que la persona que toma decisiones experimente con muchas políticas y argumentos diferentes sin cambiar o experimentar realmente con el sistema existente real.
- Permite utilizarla a la simulación como un medio de entrenamiento al personal para que obtengan experiencia en situaciones complejas.
- Actualmente lo utilizan como un medio pedagógico a la simulación.
- Por medio de la simulación podemos identificar los componentes que afectan directa e indirectamente al sistema en estudio.

El avance tecnológico en el ámbito de la computación a nivel de hardware y software; permite a la industria integrar en la línea de producción equipos y programas para su control. Como resultado directo de ello se consigue fabricar series intermedias de un producto a costos comparables a los de las grandes series y, además, la posibilidad de utilización de nuevos enfoques en la organización de la producción.

Es necesario incluir programas que en cada fase de la línea de producción gobiernen dichos equipos y permitan establecer la información necesaria para realizar operaciones de producción óptimas. Es aquí donde entran en juego los sistemas o técnicas asistidas por computadora mencionadas anteriormente. Los sistemas CAD (Diseño Asistido por computadora), CAM (Manufactura Asistida por computadora) y CAE (Ingeniería Asistida por Computadora) son los más conocidos y utilizados actualmente.

El diseño asistido por ordenador (CAD), empezó aplicándose ya en los años 60 fundamentalmente como sistema sustitutivo de los tableros de dibujo, permitiendo ganancias de tiempo en la generación de planos. Progresivamente ha ido ampliando su campo funcional de aplicación y sus prestaciones, hasta convertirse en lo que es hoy en día, una potente herramienta que permite diseñar objetos en un ordenador como si de cuerpos reales se tratase.

Los sistemas de fabricación asistida por ordenador CAM tienen por objetivo, básicamente, proporcionar una serie de herramientas que permitan fabricar el producto diseñado. Actualmente, el CAM se conoce fundamentalmente como sistema de programación "off-line" de máquinas CNC (Control Numérico Computarizado, máquinas programadas y controladas por computador a través de software de diseño y manufactura, CAD/CAM). Sin embargo, debe precisarse que el CAM es un concepto mucho más amplio, que incluye la programación de robots, de máquinas de medir por coordenadas (CMM, Máquinas de medir por coordenadas), planificación de procesos, etc.

Esta técnica nació a principios de los años 50 como una necesidad de la industria aeronáutica en procesos de obtención de parámetros de operación óptimos en naves; y es anterior a la aparición del CAD/CAM. Computer Aided Engineering / Ingeniería asistida por ordenador. [WEB-12] Son sistemas que permiten simular el comportamiento de los diferentes modelos sometidos a esfuerzos, alteraciones o cambios, movimientos, temperaturas, etc. El objetivo principal del sistema CAE al ser considerada su aplicación en los ciclos de producción de una industria, es simular el comportamiento del sistema productivo de acuerdo con las alteraciones o modificaciones de variables o factores que intervienen en el mismo, obteniéndose adicionalmente un análisis de información que permita establecer principales falencias o pérdidas en el producto final.

El grupo de técnicas de diseño asistido por computadora ha experimentado un notable progreso en los últimos años, hasta el punto de que pueden considerarse suficientemente sólidas y aplicarse de forma conveniente a lo largo de todo el proceso de diseño y fabricación de un producto. De hecho, cuando las técnicas de diseño asistido por computador se conciben de forma global e integrada el proceso se vuelve realmente efectivo.

Software de simulación en la ingeniería industrial.

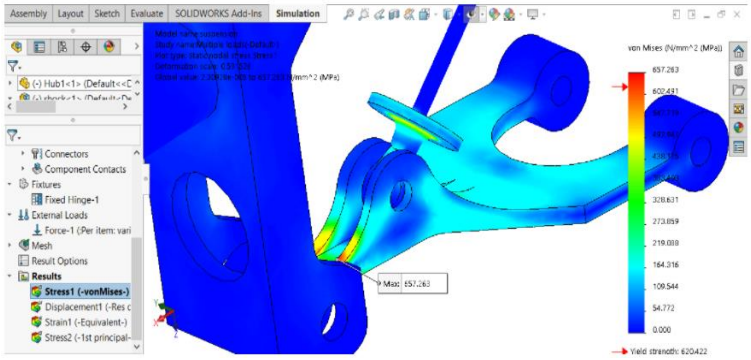
La simulación de procesos es una de las más grandes herramientas de la ingeniería industrial, la cual se utiliza para representar un proceso mediante otro que lo hace mucho más simple y comprensible. Esta simulación es en algunos casos casi indispensable, como nos daremos cuenta a continuación. En otros

casos no lo es tanto, pero sin este procedimiento se hace más complicado. Las áreas de aplicación de la simulación son muy amplias, numerosas y diversas, algunas de ellas son: Análisis del impacto ambiental causado por diversas fuentes, Análisis y diseño de sistemas de manufactura, Análisis y diseño de sistemas de comunicaciones, la simulación se utiliza en la etapa de diseño para auxiliar en el logro o mejoramiento de un proceso o diseño o bien a un sistema ya existente para explorar algunas modificaciones.

Un simulador de proceso consta de un programa ejecutivo, el conjunto de modelos matemáticos que representan las unidades de cálculo del proceso y un fichero o banco de datos con las propiedades físicas y termodinámicas de los productos que intervienen en la simulación del proceso. El tema de simuladores dedicados fundamentalmente a la industria con el objetivo de mejorar e incrementar la eficiencia de estas que permiten hacer simulaciones de diferentes procesos antes de que ocurran en realidad, las cuales producen resultados que pueden ser analizados para una futura realización de estos.

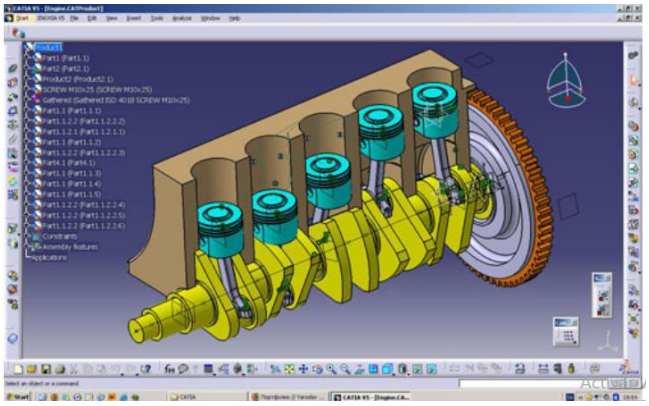
Softwares más usados:

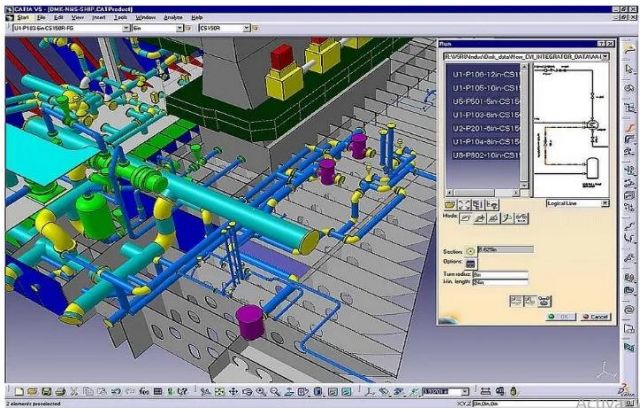
- SolidWorks. El software de diseño es tan sencillo como potente, permite que cualquier empresa pueda hacer sus ideas realidad y embarcarse en mercados globales. Las soluciones de SolidWorks se centran en la forma en que trabaja a diario, con un entorno de diseño 3D integrado e intuitivo que abarca todos los aspectos del desarrollo del producto y que ayuda a maximizar la productividad del diseño y la ingeniería. Más de 2 millones de diseñadores e ingenieros de todo el mundo usan SolidWorks para hacer realidad sus diseños desde los dispositivos más novedosos hasta innovaciones para lograr un futuro mejor. Sus diferentes módulos son: SolidWorks para diseño de producto 3D, SolidWorks simulación para análisis y simulación, SolidWorks Sustainability para diseño sustentable.



- Catvia V5. Las siglas en ingles significan computer Aided Three Dimensional Interactive Aplication, es un programa que proporciona soluciones de diseño y fabricación, está ocupando un puesto de privilegio en el modelado sólido dentro del ámbito profesional. Esta herramienta es muy importante en la industria del diseño y es uno de los programas más potentes y requeridos en el mundo por su rapidez en diseñar en 3D, es distribuido por IBM. Con este software es posible trabajar superficies avanzadas y sólidos complejos con herramientas y opciones que no poseen los

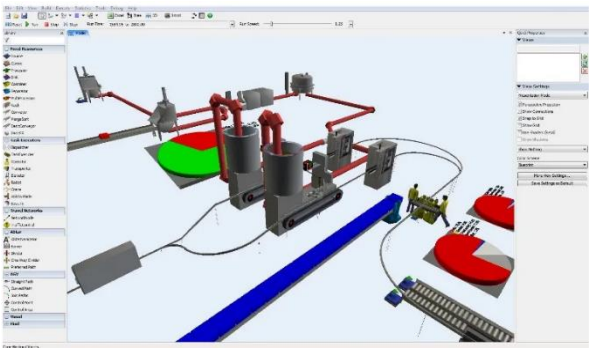
CAD de gama media. Consiste en un conjunto de aplicaciones informáticas que cubren los aspectos del diseño productivo: diseño asistido por computadora (CAD), ingeniería asistida por computadora (CAE) y fabricación asistida por computadora (CAM), con la funcionalidad necesaria para facilitar diseños industriales corporativos de todo tipo, o mediante la integración permite un apoyo continuo al proceso industrial de la empresa en su conjunto. Creado en conjunto con empresas claves industriales, Catia constituye un paso más allá en el soporte a la ingeniería integrada a la producción y puede obtener ventajas competitivas en dominios industriales, como: bienes de consumo, maquinaria industrial, sistemas para equipos industriales, construcciones navales, industria automotriz, industria aeroespacial, electricidad y eléctrica. Catia permite que los usuarios adapten su capacidad de desarrollo a sus propias necesidades, simulando la gama entera de los procesos de diseño industrial, además de resolver un amplio espectro de las necesidades del cliente.

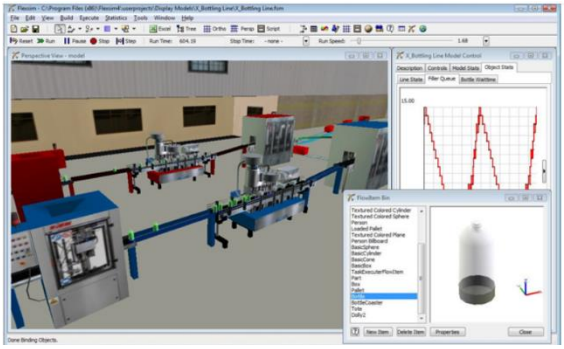




- Flexsim. Es un software para la simulación de eventos discretos, que permiten modelar, analizar, visualizar y optimizar cualquier proceso industrial, desde procesos de manufactura hasta cadenas de suministros. Es un programa que permite construir y ejecutar el modelo desarrollado en una simulación dentro de un entorno 3D desde el principio. Es usado por empresas líderes en la industria para simular sus procesos productivos, antes de llevarlo a la ejecución real. Es una herramienta de resolución de problemas que le permite responder con precisión cualquier

pregunta acerca del negocio. Es un software potente de simulación, construido desde la base para hacer la simulación lo más fácil posible, sin sacrificar una onza de función o atractivo visual. Puede crear modelos hermosos y detallados que ofrezcan resultados óptimos y se puede hacer todo en sólo minutos con controles de arrastrar y soltar, así como funciones fáciles de usar.





Bibliografía.

https://www.ms-ingenieria.com.mx/ingenieria-mecanica

La simulación y los procesos industriales by femetales - issuu

Simulación de procesos industriales | Automatización VLD 2020 (vld-eng.com)

Simulador de procesos: qué es y cuáles son sus ventajas| AUTYCOM

Sistemas de simulación: para qué sirven y dónde los aplicamos en la industria (itcl.es)

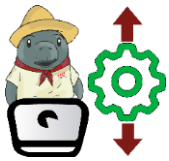
Actividad 7. Explorando el uso de simuladores industriales

Instrucciones: Después de leer la información de la lectura anterior, realiza lo que se solicita.

1. Elabora un resumen escrito donde expliques:

   - ¿Qué es la simulación y cuál es su objetivo en la industria?
   - Menciona al menos 3 beneficios clave de la simulación industrial.
   - Describe brevemente uno de los siguientes softwares: SolidWorks, Catia V5 o FlexSim.
   - Explica con tus palabras por qué es importante utilizar simuladores antes de implementar cambios en un sistema real.

2. El trabajo debe estar bien redactado y estructurado.



Práctica 2. Simulando procesos industriales con Flexim

Instrucciones: En equipo de 5 integrantes realizar la práctica simulando procesos industriales con Flexsim.

Problemática planteada.

Una planta industrial tiene un sistema de producción con un único Queue (primeras entradas primeras salidas) que alimenta cuatro estaciones de prueba en paralelo. El producto llega a la Queue cada 21 segundos desde el source (fuente). El tiempo de ciclo de pruebas es de 1 minuto. Las estaciones de prueba paran cada 20 minutos exponencialmente distribuidos y lleva entre 2 y 5 minutos uniformemente distribuidos arreglarlos. La tasa de fallos en las estaciones es del 10%. Las piezas que fallan se separan manualmente en una mesa de retrabajo, con un tiempo lognormal (35.4, 3.2, 0.1) y se vuelven a introducir al Queue que alimenta a las estaciones de prueba.

Objetos del modelo.

- El Source presenta un tiempo de arribo de 21 segundos y un flujo de primeras disponibles hacia el Queue 2 con un máximo de capacidad de contenido de 50 piezas.
- El Queue 2 presenta un flujo de producción de tipo Matching Itemtypes donde existe una separación de productos para ser asignados a los Queue 4, 5, 6, 7 respectivamente.
- Cada una de las Queues tiene una capacidad máxima de 50 productos dependiendo su tipo y presenta un flujo de proceso de tipo First Available y luego el producto es enviado hacia el área de pruebas.
- En los procesos se presentan un 10% de fallos y tiempo de proceso de 60 segundos. Se tiene un supervisor asignado para separar las piezas con defectos y volver a introducir al Queue que alimenta a las estaciones de prueba.
- Los productos que cumplan con las pruebas serán enviadas al Sink donde es la etapa final y termino del proceso.
- En caso de no ser enviados al Sink por cumplimiento de las pruebas son enviados a una mesa de retrabajo Processor 14 para su análisis y al no cumplir con las especificaciones enviadas al Queue 2 para empezar nuevamente el proceso de pruebas.

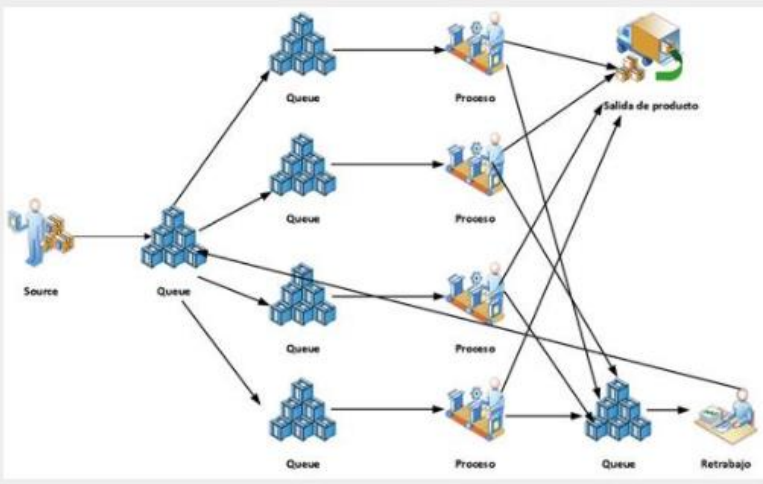


Diagrama de flujo de producción del modelo de pruebas.

Construcción del modelo.

Para construir el modelo, se inicia creando una nueva hoja de trabajo (New Model) en Flexsim. Después se selecciona de las librerías los recursos necesarios y se arrastran al área de trabajo.

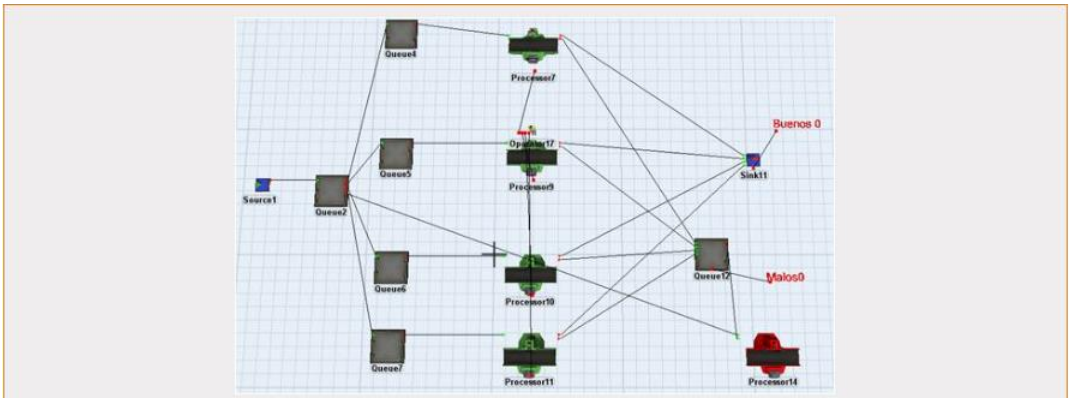


Imagen del modelo final en ejecución.

Los objetos necesarios para este caso son:

- 1 Source que es la fuente de salida de producto hacia la línea de espera Queue 2.
- 6 Queues son las líneas de espera que son los envíos hacia las operaciones o procesos.
- 5 procesos que son los encargados de las pruebas respectivas hacia los productos.
- 1 Sink que es la salida de producto terminado y que cumple con las especificaciones en forma correcta.
- 1 transporte (supervisor).

Conexión de los objetos.

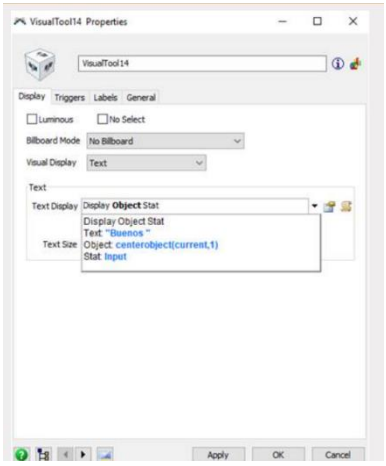
Después de arrastrar los objetos al área de trabajo, es necesario conectarlos. La conexión de objetos fluidos se hace de la misma forma con la que conecta objetos discretos: presionando la tecla A y dando clic en los objetos a conectar se crea una conexión de entrada/salida y la tecla S crea una conexión de puerto central. Hay que recordar que los objetos deben ser conectados de acuerdo con el diagrama de flujo de producción mostrado en la imagen. Las conexiones son las que se enlistan a en la tabla siguiente:

Actividad	Dependencia
Source 1	Queue 2
Queue 2	Queue 4
Queue 2	Queue 5
Queue 2	Queue 6
Queue 2	Queue 7
Queue 3	Processor 7
Queue 3	Processor 8
Queue 3	Processor 9
Queue 3	Processor 10
Processor 7	Sink 13
Processor 8	Sink 13
Processor 9	Sink 13
Processor 10	Sink 13
Processor 7	Queue 11
Processor 8	Queue 11
Processor 9	Queue 11
Processor 10	Queue 11
Queue 11	Queue 12

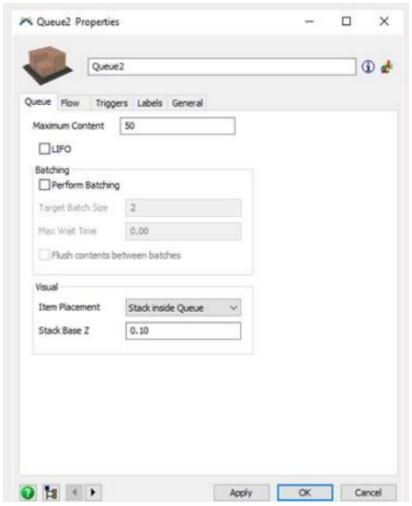
Configuración de los objetos.

- Se configura el Source, dado que el valor predeterminado del tiempo de llegadas es de 21 y el envío hacia el puerto Queue 2 es de First available.

- Configuración de las líneas de espera (Queues). Una vez posicionados los elementos de las líneas de espera se empieza la configuración dando doble clic en cualquiera de los elementos y se asigna primeramente la capacidad máxima que tendrá cada línea de espera y en este caso será de 50 cajas máximo y un envió (Send To Port) hacia las demás líneas de espera con la opción de Matching



Itemtypes ya que en el caso de la Queue 2 es la primera línea de espera de servicio de los diferentes tipos de producto a enviar a las demás líneas de espera.



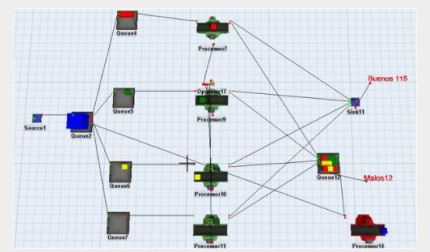
- Configuración de los procesadores (Processors). Cuando se envían los productos por las diferentes líneas de espera es cuando deben ser sometidos a las pruebas de inspección para determinar si cumplen con las especificaciones y proceder a la salida de producto terminada. Se da clic en el procesador y en la opción de Processor, se asigna una capacidad máxima de 1 caja por operación durante 10 segundos y se asigna un transporte, quien realizará los ajustes manuales o cambios en este tiempo. Recordando que se presenta un 10% de defecto, lo cual significa que los envíos al reproceso son por probabilidad.

- Configuración del Sink (salida y recepción de producto final). Después de terminar las pruebas en los diferentes procesadores y las inspecciones del supervisor los productos son enviados al Sink donde este último toma como válido el producto terminado proveniente de los procesos y es almacenado. En caso contrario que después de realizar las pruebas pertinentes envíe el producto al Queue 12, lo cual significa que no cumple con las especificaciones y es enviado al Processor 14, a la mesa de retrabajo que enviara el producto defectuoso a la línea de espera inicial del proceso de producción.

En el caso del Queue 12 y Sink 11 están programados mediante un elemento llamado VisualTool para llevar el conteo de los productos que cumplen (Buenos) y no cumplen (Malos), de acuerdo con la configuración del sistema y tener datos reales del comportamiento que tendrán y así poder tomar las mejores decisiones.

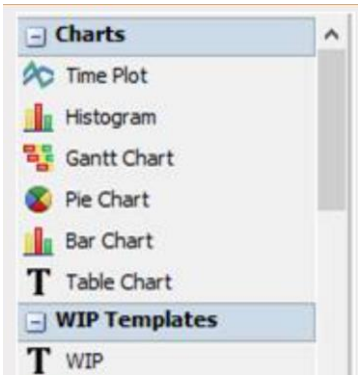
Ejecución del modelo.

Para ejecutar el modelo, se define el tiempo de simulación por ejemplo con 3600 segundos y después dar Reset una vez que el modelo esté corriendo, se observará como cada una de las líneas de espera cumple con su capacidad máxima de inicio y fin, así como los procesos en ejecución y el supervisor. Se podrá observar la cantidad total de producto aceptado y rechazado durante el tiempo total de ejecución del sistema.



Resultados.

El desempeño del sistema se ve con las estadísticas. Estas son la acumulación de valores en una variable por un tiempo determinado, inician en 0 cada vez que se restaura (Reset) el modelo y se acumulan valores hasta el momento en que se detiene la simulación. Estos resultados pueden ser vistos como números, porcentajes o gráficos, pueden programarse variables de interés para el usuario o simplemente consultarse ya definidas. Cuando se ha ejecutado el modelo durante un período de tiempo predeterminado, se pueden analizar los resultados de la ejecución en muchos y muy variadas formas.



1. Algunas estadísticas pueden ser observadas de forma rápida, las cuales se encuentran disponibles en los objetos que se muestran en el área de trabajo.
2. Se pueden agregar recursos especiales de la librería de objetos. Por ejemplo: Recorder permite añadir gráficas dinámicas y variables, las cuales se animan mientras el modelo se está simulando.

3. En el menú Statistics > Reports and Statistics se tiene acceso a un informe completo, un resumen del informe o quizá un informe del estado en el que se encuentra el modelo.

No olvidar que, para hacer una inferencia válida, los modelos de simulación se estabilizan haciendo ejecuciones con períodos de tiempo largos. Es importante tener en cuenta la importancia de ejecutar el número necesario de réplicas, debido a que las ejecuciones múltiples contribuyen también a mejorar la validez de la inferencia. Un mayor número de réplicas implica un número más elevado de muestras aleatorias independientes con distribución estadística diversa.

Conclusiones.

Contesten lo siguiente: ¿Cómo operan los objetos en Flexsim? ¿La simulación en Flexsim permite tomar mejores decisiones en la operación de los sistemas industriales? ¿Por qué es importante la simulación en la industria?

Bibliografía.

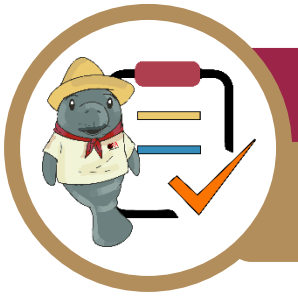
Acosta Flores, Ingeniería de sistemas: un enfoque interdisciplinario. 2a ed., Centro de Investigaciones Interdisciplinarias en Ciencias y Humanidades de la UNAM, México: AlfaOmega, 2007, pp. 1-26.

https://www.redalyc.org/jatsRepo/614/61458109002/html

M. Beaverstock, A. G. Greenwood, E. Lavery, W. Nordgren, Applied Simulation Modeling and Analysis using FlexSim, 3a ed., FlexSim Software Products, Inc., Orem USA, 2012.

M. Barceló, Una historia de la informática, Rambla del Poblenou, Barcelona: UOC, 2008, pp. 77-80.

R. Coss-Bú, Simulación: un enfoque práctico, 20a ed., Noriega/Departamento de Ingeniería Industrial, Instituto Tecnológico y de Estudios Superiores de Monterrey/Limusa, México, 2003, pp. 11-18.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 6

Práctica 2: Simulando procesos industriales con Flexim

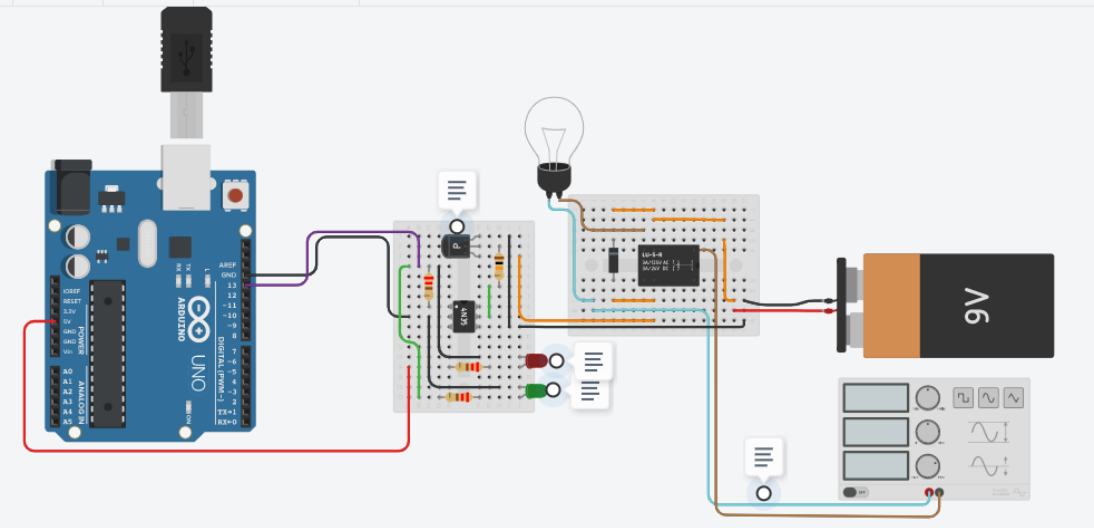
DATOS GENERALES		
Nombre(s) del alumno (s)		**Matriculas(s)**
Producto: Reporte de la práctica		**Fecha**
Formación laboral básica: Robótica **Submódulo 2:** Control industrial		**Periodo: 2026-2027A**
Nombre del docente:		**Firma del docente**

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Ingresa a Flexsim.			1		
2.	Inserta los objetos del modelo.			2		
3.	Construye el modelo con los objetos indicados.			2		
4.	Realiza la conexión y configuración de los objetos.			2		
5	Ejecuta el modelo con el tiempo señalado y realiza las estadísticas solicitadas.			2		
6	Contesta las preguntas de la conclusión en forma entendible.			1		
				CALIFICACIÓN		

Ejercicio 7. Modelo de potencia en Tinkercad

Instrucciones: Realiza el Ejercicio 7 creando el circuito en potencia en Tinkercad.

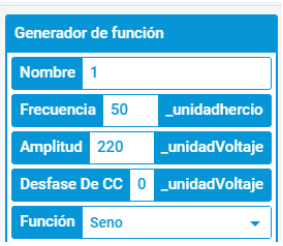
Circuito.



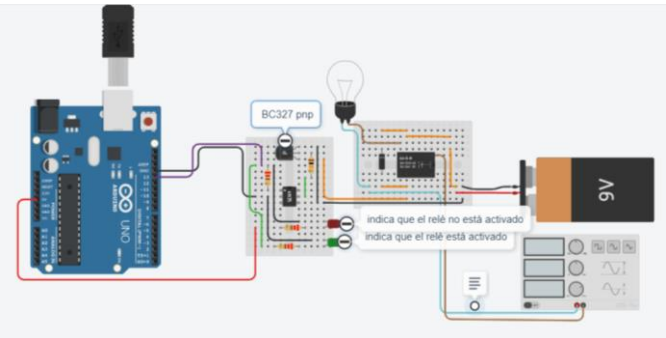
Componentes.

- Arduino Uno R3.
- 2 miniplacas de prueba.
- 1 bombilla.
- 1 relé SPDT.
- 1 transistor PNP (BJT).
- 1 optoacoplador.
- 1 diodo.
- 3 resistencia de 220 Ohm.
- 1 resistencia de 1 Kohm.
- 1 led rojo.
- 1 led verde.
- 1 batería de 9 Volts.

- Cables en color: naranja, rojo, verde, negro, morado, azul claro, café.
- 1 generador de frecuencia 50 hercios, amplitud 220 volts, desfase de CC de o volt y la función seno



Incluir las siguientes notas en el circuito.



- BCP327 pnp
- Indica que el relé no está activado.
- Indica que el relé está activado
- 220 VCA.

Código.

```
void setup()
{
  pinMode(13, OUTPUT);
}
void loop()
{
  digitalWrite(13, HIGH);
  delay(3000); // Wait for 1000 millisecond(s)
  digitalWrite(13, LOW);
  delay(3000); // Wait for 1000 millisecond(s)
}
```

Lectura 8. Dibujo industrial

Un dibujo es una delineación o un trazo, por lo general se realiza manualmente con la ayuda de un pincel, un lápiz u otra herramienta con el objetivo de representar una figura o una idea. El dibujo industrial por su parte es aquello vinculado a la industria: las instalaciones y los procesos que permiten obtener, transformar y comercializar productos naturales o materias primas.

Existen múltiples tipos de dibujos según sus características. Podemos decir que los dibujos industriales son aquellos gráficos o planos que brindan información de utilidad sobre un procedimiento o un dispositivo de un sector de la industria.

El dibujo industrial permite plasmar ideas y comunicar proyectos a través del uso de escalas, perspectivas y diversas técnicas de representación. En estos dibujos se utilizan símbolos para facilitar la inclusión de datos que sean fáciles de comprender para todos los profesionales. La idea de dibujo industrial se vincula al concepto de diseño industrial, el cual es el proceso creativo que permite describir y representar la configuración de un elemento que puede servir para mejorar las cualidades de algún producto. El diseño industrial se puede realizar de manera tridimensional o bidimensional.

Cuando el diseño es 3D, se habla de un modelo industrial, se trata de un objeto que ocupa un lugar espacial. Cuando es en 2D se desarrolla en un plano en donde combina líneas y colores.

Con el desarrollo de un dibujo industrial es posible disponer, reunir y combinar figuras, como son líneas y colores en un espacio plano con el objetivo de utilizarlos para ornamentar un producto industrial y proporcionarle una nueva apariencia. El desarrollo de un producto tiene varias fases y la estética es una de las últimas.

La diferencia principal entre diseño y dibujo industrial reside en la fase del desarrollo. Cualquier patrón que se pretenda usar para estampar telas, géneros o materiales laminares de un producto se engloban en el conjunto de dibujos industriales, con la condición de que pueden ser calificados de novedosos.

Al momento en que inicia el proceso de creación de un producto industrial, lo normal es que el fabricante se enfoque en el aspecto funcional del mismo, en la razón por la cual ha decidido crearlo. Debe encontrar un equilibrio entre las características que desee incluir y las posibilidades que el mercado le proporcione para alcanzar esa meta, debido a esto el resultado final no siempre corresponde al 100% con las expectativas iniciales. De nada sirve la apariencia física antes de conocer la forma que tendrá el producto. No se comienza por todos los detalles de tipo ornamental, aquellos que permiten al consumidor distinguirlo d

ellos demás sin necesidad de conocer sus funciones o sus características principales. Parece que esto fuera superficial y técnicamente lo es, en un mundo competitivo y gobernado por la economía es un punto absolutamente necesario.

El dibujo industrial permite caracterizar un producto, darle una identidad estética que se debe complementar con la que de por si le brinda su diseño, sus prestaciones, sus innovaciones tecnológicas o su precio accesible, entre otros rasgos. Algunas de las cuestiones que la persona a cargo del dibujo industrial debe plasmar en él son las dimensiones de las diferentes partes, la forma y los materiales que se usarán en la fabricación. Esto se logra por medio de croquis y esquemas.

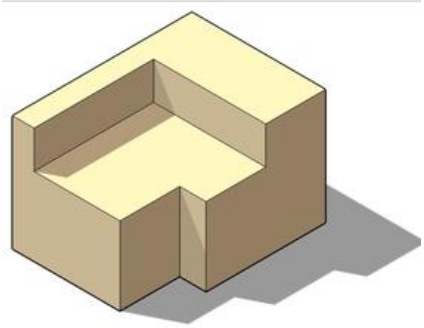
Vistas.

La croquización se conoce como vistas de las pieza u objeto a la imagen de este que se observa desde una determinada posición, en donde podemos decir que en las piezas industriales hay tres tipos de planos con formas definidas. La observación y el estudio de estos planos facilita enormemente el trabajo que se tenga realizar posteriormente. Estos planos son:

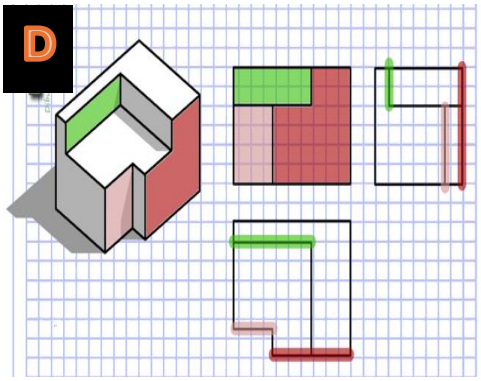
- Planos paralelos y perpendiculares a los planos de proyección.
- Planos oblicuos a los planos de proyección.
- Planos circulares.

Todos ellos, pueden ser planos vistos y planos ocultos. Se mostrará una pieza con planos paralelos y perpendiculares a los planos de proyección.

Representación.

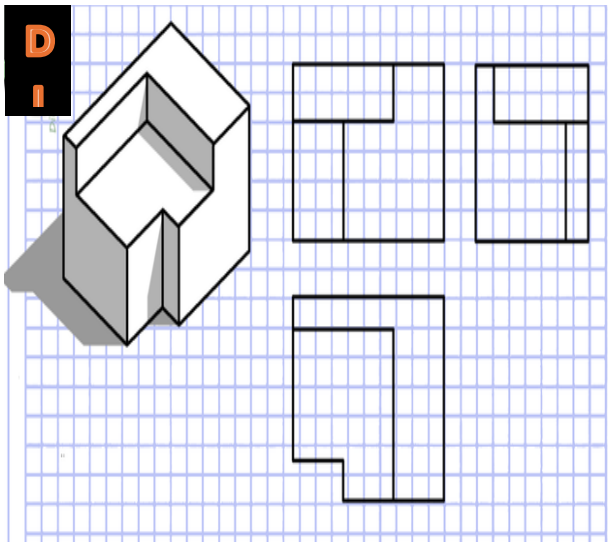


En forma general se trata de piezas muy simples. Para facilitar la tarea de representación, se pueden utilizar colores que ayuden a diferenciar los distintos planos y ver cómo se comportan. La imagen con los planos coloreados se vería así:

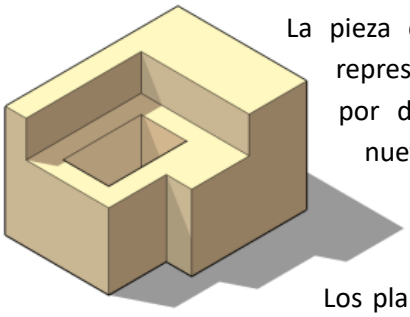


El plano verde es paralelo al plano vertical y es perpendicular a los planos horizontal y de perfil. Así, el plano verde se verá representado en el alzado con la misma forma y tamaño, mientras que en la planta y en el perfil, se verá como una línea representada en verde. El plano rojo y el rosa son paralelos al plano verde y debido a esto se representan de la misma forma que el plano verde. El resto de los planos

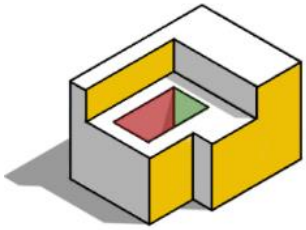
no coloreados se comportarán de la misma forma. Lógicamente los colores se pueden utilizar en el periodo de aprendizaje. La representación final se tiene que hacer sin los colores, como se ve a continuación:



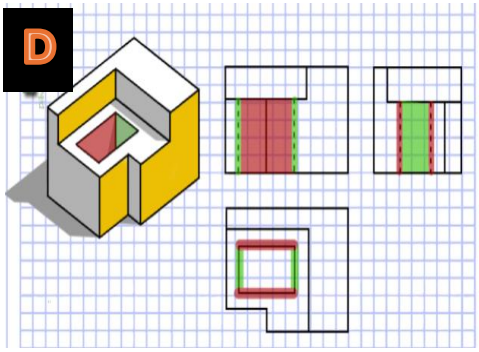
Planos paralelos y perpendiculares a los planos de proyección, pero con zonas ocultas.



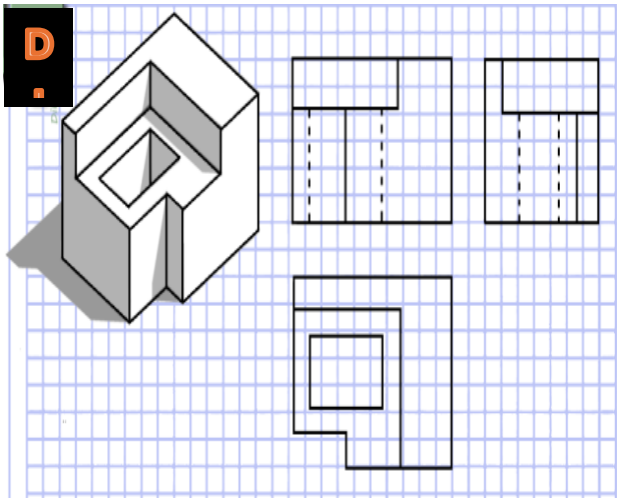
La pieza es similar a la anterior y muy simple de representar, pero en este caso se complica un poco por disponer de una zona oculta. Se utilizarán nuevamente los colores para analizar cada uno de los planos que nos aporten información relevante para la representación de la pieza.



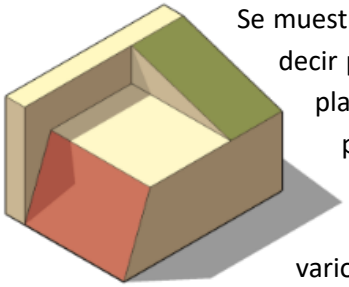
Los planos amarillos son los planos paralelos y perpendiculares a los planos de proyección. Esta pieza es diferente a la anterior porque tiene un agujero pasante que atraviesa toda la pieza, con dos planos paralelos y perpendiculares a los planos de proyección, dibujados con los colores rojo y verde. Los planos rojos son paralelos al plano vertical, por lo que se ve representado en el alzado con la misma forma y tamaño, mientras que estos mismos planos son perpendiculares a los planos horizontal y de perfil, por lo que se verán como una línea en la planta y en el perfil.



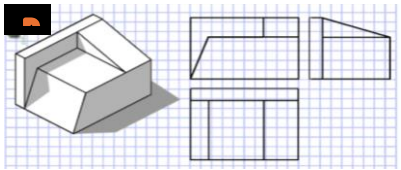
El plano verde y su paralelo, es paralelo al plano de perfil y perpendicular a los otros dos, por lo que ve en su verdadera forma y tamaño en el perfil, mientras que en la planta y el alzado se ve como línea. Los planos ocultos se representan con línea de trazos o discontinua, mientras que los planos vistos se representan con líneas llenas y continuas. La representación será igual que la anterior, pero añadiendo los elementos relativos al agujero. De nuevo se debe prescindir de los colores, por lo que quedaría de esta forma:

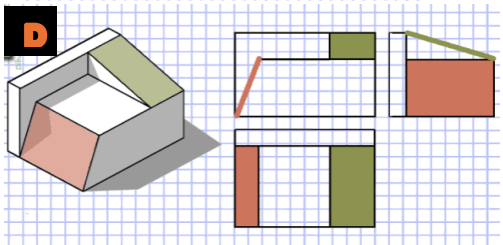


Planos oblicuos.



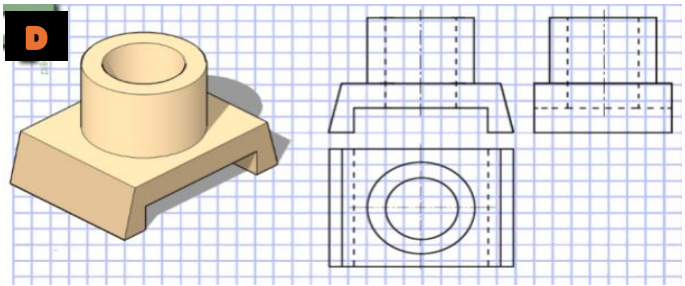
Se muestra una pieza con bastantes planos, es decir planos paralelos y perpendiculares a los planos de proyección. Existen unos planos distintos, son los planos oblicuos, los cuales son planos inclinados con respecto a uno o varios planos de proyección y que aparecen coloreadas en verde y rojo. En la pieza del ejemplo, se ve que el plano rojo es perpendicular al plano vertical, por lo que se ve como una línea en el alzado y el oblicuo es decir inclinado con respecto a los planos horizontal y de perfil. En la planta como en el perfil, este plano oblicuo se ve con una forma y un tamaño distinto del real. El plano verde es parecido, es perpendicular al plano de perfil por lo que nos da una línea como proyección en el perfil y es oblicuo con respecto al plano horizontal y vertical. Este plano tendrá una distorsión en la planta y el alzado. Necesitamos representar la pieza de forma concreta, es decir sin colores, la cual queda de la siguiente forma:





Planos circulares.

Se trata de una pieza que tiene todos los tipos de planos, son piezas comunes, con mayor o menor complejidad, que podemos encontrar en la industria. Analizando la pieza, se puede ver que tiene un agujero que atraviesa toda la pieza. Por ello es necesario representar esas



zonas ocultas. Las líneas utilizadas en la representación de las piezas (vistas), deben tener unas características determinadas que son las líneas normalizadas.

Bibliografía.

Calderón Barquín, Francisco José. (2017). Dibujo técnico industrial. Editorial Porrúa. ISBN: 9789700766188

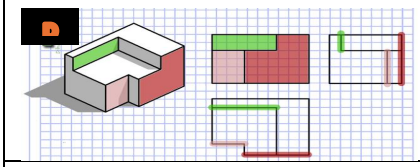
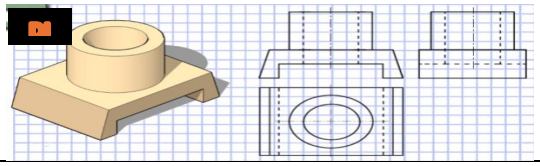
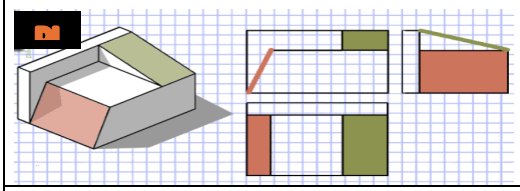
Ibáñez Carabantes, Pedro, Ubieto Artur, Pedro, Auria Apilluelo, José Manuel. (2005). Dibujo industrial. Conjuntos y despieces. Editorial Parainfo. ISBN: 9788497323901

https://ibiguridt.wordpress.com/2020/05/07

Vistas de piezas | Dibujo Técnico (wordpress.com)

Actividad 8. Dibujo industrial

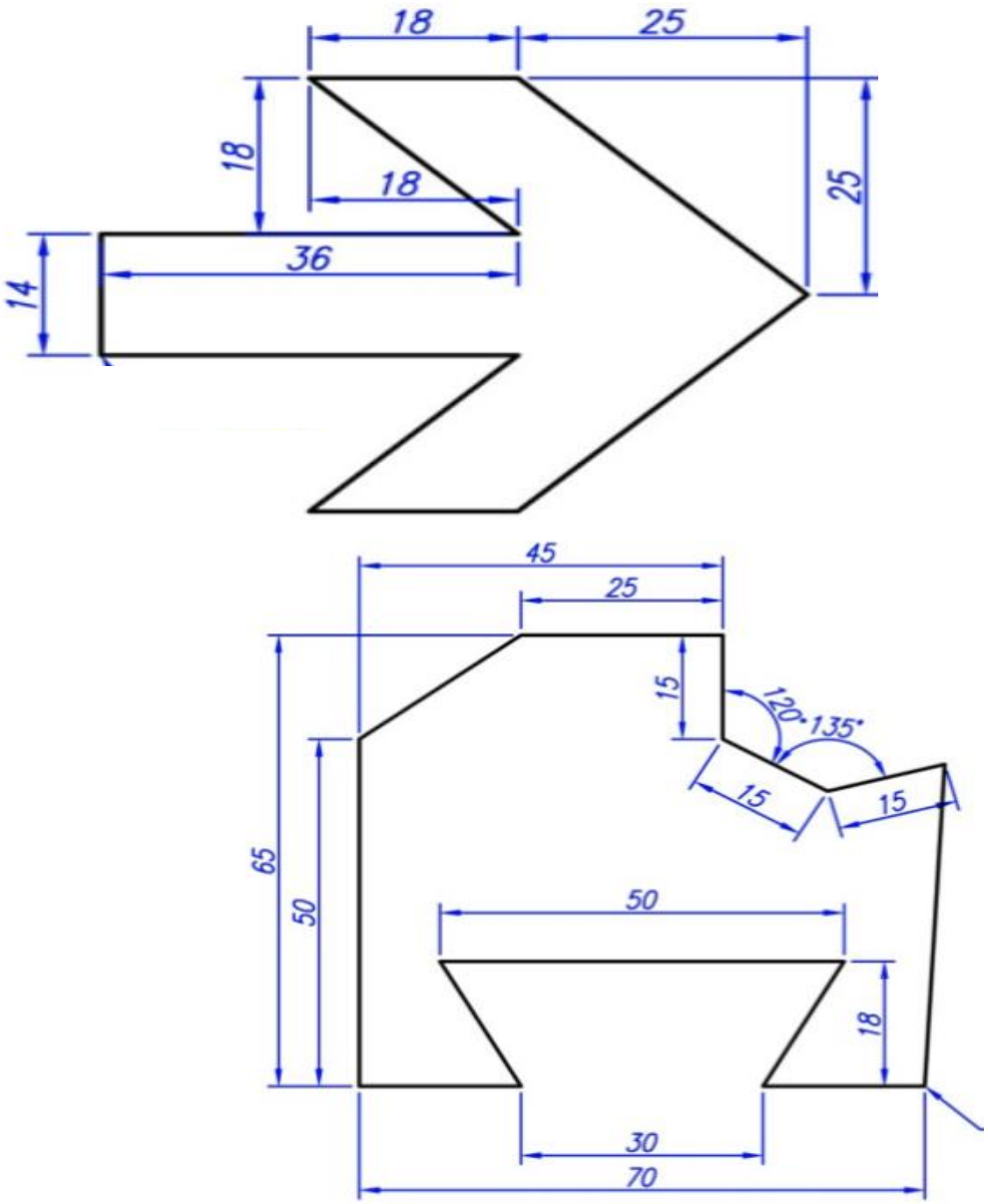
Instrucciones: Después de realizar la Lectura 8 relaciona las columnas.

1.Es una delineación o trazo	( ) Modelo industrial
2. Está vinculado a las instalaciones y procesos	( ) Plano paralelos y perpendiculares a los planos de proyección
3. Sirven para facilitar la inclusión de datos que sean fáciles de comprender para todos los profesionales	( ) Vistas
4. El diseño 3D es	( ) Planos circulares
5. El diseño 2D es	( ) Dibujo
6. La croquización es	( ) Planos oblicuos
7. 	( ) Croquis y esquemas
8. 	( ) Dibujo industrial
9. 	( ) Identidad estética
10. Permite caracterizar un producto, darle	( ) Plano
11. Las dimensiones de las diferentes partes, forma y los materiales que se usarán en la fabricación, se logra por medio de	( ) símbolos

Ejercicio 8. Vistas y croquis en papel

Instrucciones: Realiza el Ejercicio 8 de vista de planta y el croquis en papel.

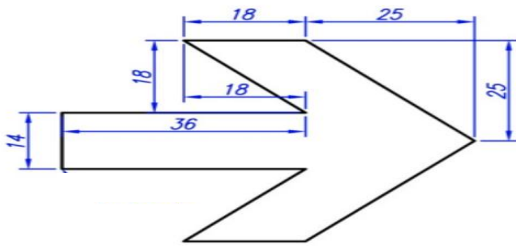
Imagen en vista de planta.

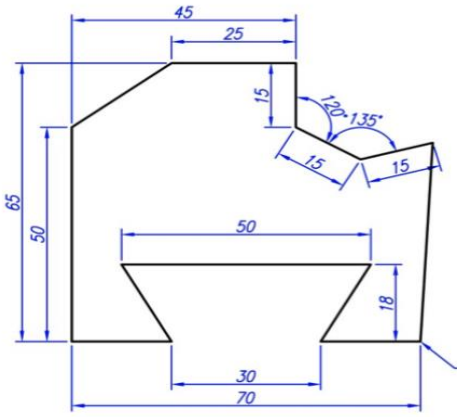


Ejercicio 9. Vistas y croquis en Autocad

Instrucciones: Realiza el Ejercicio 8 de vista de planta y el croquis en Autocad usando los comandos Línea y cotas.

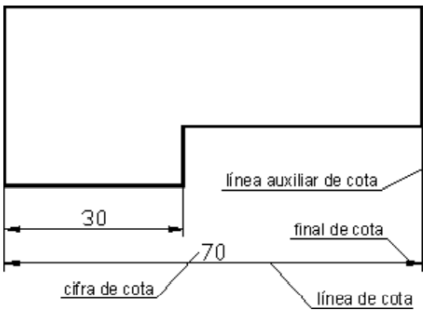
Imagen en vista de planta.



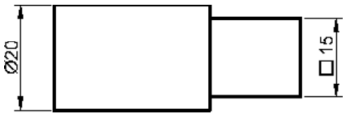


Procedimiento.

1. Ingresar a Autocad.
2. Para establecer el formato y la precisión de las unidades. En el menú formato seleccionar Unidades En el cuadro de diálogo Unidades de dibujo, en Longitud seleccionar el formato de unidad y la precisión. El área muestra de salida mostrará un ejemplo del formato de unidad con la precisión actual. Dar clic en Aceptar. Cuando se hace un dibujo en papel se debe determinar la escala antes de iniciar el dibujo. Esta escala compara el tamaño del objeto dibujado con el tamaño real del objeto representado por el dibujo. En Autocad el usuario dibuja con un tipo de unidad especificado que por defecto es decimal. Cada unidad de la pantalla representa lo que el usuario desee, una pulgada, un milímetro, un kilómetro, etc., por ejemplo, si se dibuja una parte de un motor, una unidad puede ser equivalente a un milímetro y si se dibuja un mapa una unidad puede ser equivalente a un kilómetro. Cuando se establece el factor de escala, se puede utilizar para definir la altura del texto, los tamaños de acotación y las escalas de tipo de línea, de patrones de sombreado y de ventana gráfica. Terminado el dibujo se puede trazar a cualquier escala o con diferentes vistas, cada una a una escala distinta.



3. Se considera que el dibujo de una pieza o mecanismo está correctamente acotado cuando las indicaciones de cotas utilizadas sean las mínimas, suficientes y adecuadas, para permitir la fabricación de esta. Las cotas se sitúan por el exterior de la pieza, se admiten en el interior cuando no se pierde claridad en el dibujo. Los



elementos básicos que intervienen en la acotación son: la parte de la línea de referencia donde se rotula el texto, se dibuja paralela al elemento a acotar, si este no queda bien definido, se dibuja en forma horizontal o sin línea de apoyo para el texto.

4. Dar clic en la ficha Inicio > grupo Dibujo > Línea . Especificar el punto inicial y final del segmento de línea. Dar clic en el área de dibujo. Se continúa especificando segmentos de líneas adicionales, para deshacer el segmento de línea anterior, escribir h en la solicitud de comando. Dar clic en Deshacer en la barra de herramientas de acceso rápido para canelar toda una serie de segmentos de línea. Dar Intro o Esc cuando se haya terminado o escribir c para cerrar una serie de segmentos de línea.

5. Para líneas con coordenadas específicas, se da clic en la ficha Inicio > grupo dibujo > Línea escribir el valor de la coordenada del primer punto, para ello se especifica el valor de X, una coma y a continuación el valor de Y, por ejemplo: 1.65,4.25. Se presiona la barra espaciadora o la tecla Intro. Se puede optar por alguna de las siguientes acciones: si la entrada dinámica está activada, se escribe el signo de numeral (#), seguido del valor de X, una coma, a continuación, el valor de Y, por ejemplo: #4.0,6.75. Si la entrada dinámica está desactivada, escribir el valor de X, una coma, a continuación, el valor de Y, por ejemplo: 4.0,6.75. Presionar la barra espaciadora o la tecla Intro. Se puede presionar la tecla F12 para activar o desactivar la entrada dinámica.
6. Para dibujar una línea con una longitud específica, se da clic en la ficha Inicio > grupo Dibujo > Línea . Designar el punto inicial, para especificar la longitud se puede hacer lo siguiente: desplazar el cursor para indicar la dirección y el ángulo, introducir la longitud, por ejemplo: 6.5. También se puede introducir una arroba (@) y la longitud, después escribir un signo menor que (<) y el ángulo, por ejemplo: @6.5<45. Presionar la barra espaciadora o la tecla Intro.

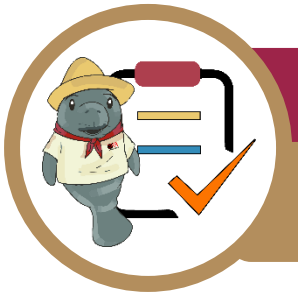
7. Para dibujar línea con un ángulo específico en otra línea. La alineación temporal del icono SCP con una línea existente facilita el dibujo de línea. En la solicitud de comando continuación, seleccionar la línea (0,0,0). Se da clic en la ficha Inicio >  una línea con un ángulo específico en otra escribir SCP, escribir ob para objeto y a existente. Se define el origen del SCP grupo Dibujo > Línea . Designar el punto inicial. Se puede realizar alguna de las siguientes acciones para especificar el ángulo: introducir el signo menor que (<) y el ángulo, por ejemplo: <45 y desplazar el cursor para indicar la dirección, o desplazar el cursor para indicar el ángulo aproximado. Designar el segundo punto. Presionar la barra espaciadora o la tecla Intro. En la solicitud de comando, escribir SCP, escribir p para Anterior a fin de restablecer el origen del SCP.

8. Para dibujar una línea con un ángulo específico. Dar clic en la ficha Inicio > grupo Dibujo > Línea . Designar el punto inicial. Se puede hacer lo siguiente: introducir el signo menor que (<) y el ángulo, por ejemplo: <45 y desplazar el cursor para indicar la dirección. Introducir las coordenadas polares, por ejemplo: 2.5 <45. Pulsar F8 para activar el modo Orto a fin de bloquear el ángulo en las direcciones horizontal y vertical. También se puede pulsar Mayús al especificar el siguiente punto de una dirección horizontal o vertical. Pulsar F10 para activar el rastreo polar. Es posible que se deba usar el comando PARAMSDIB, ficha seguimiento Polar, para especificar ángulos polares adicionales y elegir la opción para rastrear todos los parámetros de ángulo polar. Desplazar el cursor para indicar un ángulo aproximado. Se puede realizar alguna de las acciones siguientes para especificar la longitud: dar clic en un punto para especificar el punto final con o sin referencias a objetos, recorte o alargue la línea resultante como sea necesario, o se puede especificar la longitud de la línea, por ejemplo: 2.5. Presionar la barra espaciadora o la tecla Intro.
9. Realizar los dos dibujos con las medidas que se solicitan, así como las cotas especificadas.

Bibliografía.

Manual de Estudio (ipn.mx)

Para dibujar líneas | AutoCAD 2020 | Autodesk Knowledge Network



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 7

Ejercicio 9: Vistas y croquis en Autocad

DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Dibujos en Autocad		**Fecha**	
Formación laboral básica: Robótica **Submódulo 2:** Control industrial		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Ingresa a Autocad.			1		
2.	Utiliza el grupo Dibujo con comando línea.			2		
3.	Crea el primer dibujo con la forma de la flecha.			2.5		
4.	Crea el segundo dibujo con la forma que se especifica.			2.5		
5	Crea las cotas en cada uno de los dibujos			2		
				CALIFICACIÓN		

Lectura 9. Dibujo industrial en Autocad

En la realización de cualquier dibujo para un proyecto es necesario la elaboración de presentaciones e impresiones de planos que proporcionen la información necesaria, para entender el funcionamiento o producción, de igual forma se debe recordar que cuando se utilizan símbolos gráficos, el lenguaje universal del dibujo técnico permite la lectura e interpretación de estos. La creación de dibujos en AutoCAD solo es un paso en la cadena de producción, la fase de dibujo solo es la conceptualización del proyecto. AutoCAD permite configurar el dibujo de tal forma que pueda dar una noción previa de la versión final. Es un software especializado de dibujo técnico, ofrece la posibilidad de imprimir los dibujos realizados en formatos estandarizados que posteriormente se producen, es decir, generar planos, esquemas y proyecciones de objetos, instalaciones, edificios, detalles de piezas mecánicas y dimensiones.

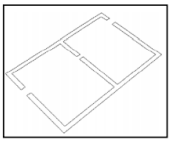
El dibujo técnico, también se conoce como dibujo de ingeniería, es un diagrama o un plano detallado y preciso que transmite información sobre cómo funciona o se construye un objeto. Ingenieros, electricistas y contratistas utilizan estos dibujos como guías al construir o reparar objetos y edificios. Los dibujos técnicos sirven de puente de comunicación entre los diseñadores y las personas que aportan ideas, los productores y las personas que ponen en práctica esas ideas. Se han diseñado como un lenguaje universal que entienden ingenieros, contratistas y arquitectos.

La administración de estilos se emplea, para mostrar diferentes cualidades de un objeto realizado, siendo de vital utilidad al momento de crear las presentaciones finales y mostrar diversos aspectos del diseño durante su creación. En AutoCAD existen diferentes administraciones de estilo:

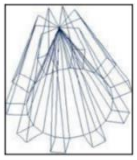
1. Estilo visual. Es un grupo de parámetros que controlan la visualización de aristas y de sombreados en la ventana gráfica. En lugar de utilizar comandos y de configurar variables de sistema, puede cambiar las propiedades del estilo visual. Cuando se aplique un estilo visual o se cambie los parámetros, se puede ver el efecto en la ventana gráfica. AutoCAD contiene estilos visuales que permiten previsualizar un dibujo:

- Conceptual. Con este estilo se añaden sombras y a la vez se suavizan los bordes entre las caras que lo conforman. En esencia es similar al estilo oculto 3D, pero con la edición de colores degradados para darle un aspecto más "realista" a la visualización, sin necesidad de asignar algún material para el modelizado (render).

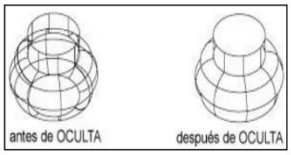
- Estructural alámbrica 2D. En esta se presentan los objetos como líneas o curvas, como cualquier dibujo en AutoCAD, este estilo es el que se muestra predeterminado al iniciar AutoCAD, siendo la opción en la que se trabaja si es que no se configura en algún momento.



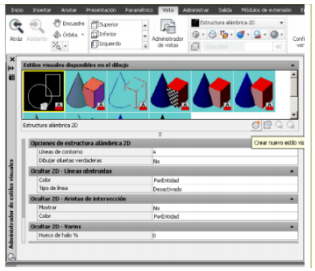
- Estructural alámbrica 3D. al colocar esta opción, AutoCAD visualiza la estructura alámbrica como se observaría en un dibujo 3D.



- Oculto 3D. Es similar a la estructura alámbrica en 3D, solo que las líneas de las caras ocultas no se mostrarán.



- Realista. De forma similar al conceptual, se agregan sombras, así como suavizado en los bordes de las caras de los objetos, pero adicionalmente, permite la visualización de los materiales que han sido asignados al objeto.



2. Estilo de trazado. Se administra a los objetos realizados, estando directamente inferido al "trazo". Se aplica desde las capas o en la presentación de impresión. Con las propiedades de estilo de trazo, es posible modificar lo siguiente:

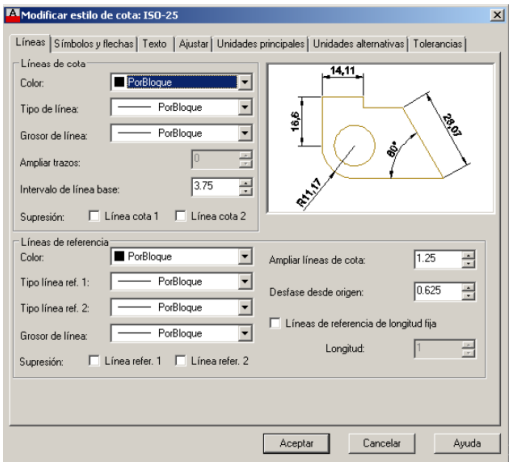
- Color.
- Simulación de color.
- Escala de grises.
- Número de plumilla.
- Plumilla virtual.
- Tramado.
- Tipo de línea.
- Grosor de línea.
- Transparencia.
- Estilo de final de línea.
- Estilo de junta de línea.
- Estilo de relleno.

Para administrar el trazo de los objetos realizados en las presentaciones es necesario asignar una tabla diferente para cada presentación del o los dibujos creados. Para visualizar el efecto de una tabla de estilos

trazados sobre una presentación, en el cuadro de diálogo, Configuración de página, en Tabla de estilos trazado. Seleccionar Mostrar estilos de trazado. El procedimiento para asignar una tabla de estilo de trazado:

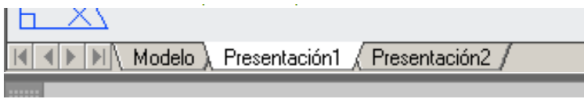
- Dar clic en la ficha Modelo o en la ficha de presentación a la que desees asignar la tabla de estilos de trazado.
- Dar clic en la ficha de salida panel Trazar Administrador de configuraciones de página.
- En el administrador de configuración de página, dar clic en modificar.
- En tabla estilos trazado (asignación, plumillas), seleccionar una tabla de estilos de trazado de la lista.
- En el cuadro de diálogo pregunta dar clic en SI o NO para indicar si la selección se debe aplicar sólo a la ficha actual o a todas las presentaciones. Esta opción sólo se encuentra disponible en la ficha Modelo.
- Para previsualizar los efectos de la tabla de estilos de trazado en la presentación, se da clic en Mostrar estilos de trazado. Esta opción está disponible solamente para presentaciones.
- Dar clic en Aceptar.
- En el Administrador de configuraciones de página, dar clic en cerrar. Si la opción Mostrar estilos de trazado estaba activada en el cuadro de diálogo se debe escribir regen en la solicitud de comandos para mostrar los parámetros del estilo de trazado. El procedimiento para previsualizar los efectos de una tabla de estilos en una presentación: Dar clic en la ficha presentación en la que se desea previsualizar los efectos de la tabla de estilos trazados. Dar clic en la ficha de Salida Administrador de configuraciones de página. En el Administrador de configuraciones de página dar clic en Modificar. En el cuadro de diálogo Configuraciones de página, en Tabla estilos trazado (asignado plumillas), seleccionar la opción Mostrar estilos de trazado. Dar clic en aceptar. En el administrador de configuraciones de página, dar clic en Cerrar. En la presentación se podrá ver el efecto de la tabla seleccionada.

3. Estilo de cota. Al realizar un dibujo en AutoCAD por muy simple que sea, requiere contener cierta información, para su instalación o elaboración, el uso de acotaciones permite conocer las dimensiones reales del objeto. Para la acotación se emplean los aspectos formales de la acotación, el texto, flechas y líneas de cota. Para acceder al estilo de cota: Formato > Estilo de cota. Cuando se ingresa al comando y se abre esta ventana, se gestionan los estilos de acotado; se puede agregar un nuevo estilo, modificar, reemplazar el actual o comparar diferentes estilos. En la carpeta de línea se configuran las extensiones de la cota, el color, tipo y grosor de línea, así mismo se puede agregar una distancia diferente entre la cota y el dibujo. En símbolos y flechas se configura la forma del extremo de la cota y el tamaño de esta. En la carpeta de texto se modifica el tamaño, estilo de texto y color. En la ficha ajustar se activa la ventana para visualizar todos los elementos de la acotación,



permitiendo hacer modificaciones generales. En la opción de unidades se administra el formato (m, mm, pulgada, etc.) y supresión o agregado de ceros para la cifra numérica.

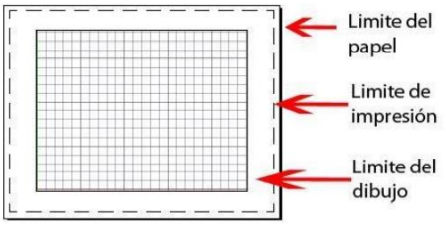
Administración de presentaciones.



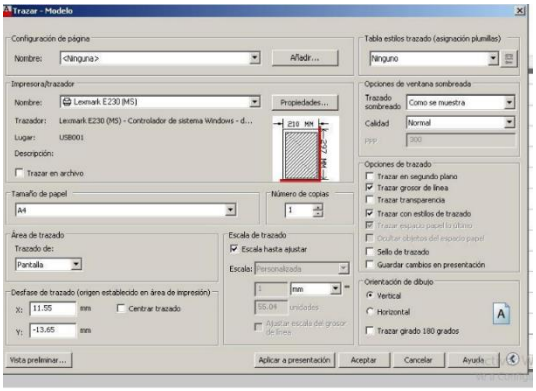
La administración de presentaciones o papel se emplea al momento de preparar el dibujo para impresión, es decir que se realizan los detalles finales de acomodo, para la presentación adecuada del plano sobre la hoja. Cuando se inicia AutoCAD automáticamente el dibujo se realiza en la presentación de "Modelo", pero para presentar nuestro dibujo de forma adecuada es importante colocarlo en la opción "Presentación" o Layout.

La pestaña de presentación contiene dos opciones "Papel" y "Dibujo". En la opción de papel es posible modificar el tamaño del papel de impresión y el margen, en dibujo se puede mover el plano en la posición deseada. Al observar la imagen nos damos cuenta de que al activar la opción de presentación se cambia de ventana donde el fondo es blanco y se muestra el límite de papel, límite de impresión y límite del dibujo, esto sirve para acomodar en forma adecuada en la hoja de impresión. La opción de administración de presentaciones es el paso previo para la impresión, en esta fase del proceso se configura la salida del archivo, esta puede ser como archivo PDF o la impresión del plano en los formatos establecidos por la normalización.

Después de la opción de presentación, es posible administrar la configuración de página, esta opción es indispensable para las propiedades de la impresión. Para configurar el tamaño del papel para impresión se realiza lo siguiente: Clic derecho en la pestaña de presentación > administrador de configuración de página. Cuando se abre esta ventana, es necesario conocer los siguientes conceptos para administrar adecuadamente las configuraciones.



- Tipo de salida. Puede ser física, en un plano en papel. En este caso se necesita configurar el tipo de impresora, tamaño de papel, etc. También puede ser electrónica, como en un archivo de dibujo en AutoCAD, para internet tipo DWF, archivos de dibujo como JPG, PNG, o un archivo de impresión tipo PLT.
- Unidades de dibujo. Es el momento en que entra en juego las unidades de nuestro dibujo, si está en milímetros, pulgadas, etc., para indicárselo a AutoCAD.
- Escala del dibujo. Relación entre la medida real y la presentada en el papel. Por ejemplo, una escala 1:10 significa que una unidad del plano representa 10 en la "vida real".



- Al activarse esta ventana se puede modificar la impresora, el tamaño del papel y la escala, así como la calidad y posición de la hoja.

Bibliografía.

La nube artística. (n.d.). Normalización. El proyecto: Normalización. Formatos, escritura, líneas y vistas.

Uso de tablas de estilos de trazado de AutoCAD | AutoCAD | Autodesk Knowledge Network

López Lucas, B. (2012). Dibujo técnico. Obtenido de: http://www.dibujotecnico.com/

Poveda, R. (2008). Interpretación de planos MMT12. Antofagasta, Chile: Universidad Antofagasta.

Sánchez, A. (2015). Cómo imprimir en AutoCAD en Espacio Modelo. Valores de impresión. Obtenido dehttps://www.andresdeltoro.es/como-imprimir-en-autocad-en-espacio-modelo/

Servicio Hidráulico Industrial. http://www.serviciohidraulico.com.mx/simbologia-hidraulica.html

Actividad 9. Explorando los estilos y presentaciones en Autocad

Instrucciones: Lee cuidadosamente la información sobre los estilos y administración de presentaciones en Autocad (Lectura 9) y realiza lo que se solicita.

Elabora un documento (escrito a mano o digital) que contenga:

Parte A – Estilos en AutoCAD (4 puntos)

- Define brevemente qué es un estilo visual, un estilo de trazado y un estilo de cota.
- Menciona un ejemplo práctico de cómo usar cada uno de esos estilos en un plano técnico.

Parte B – Administración de Presentaciones (3 puntos)

- Explica con tus palabras qué es la presentación (Layout) y por qué es importante para la impresión de planos.
- Menciona tres configuraciones clave que deben realizarse en la presentación antes de imprimir.

Parte C – Reflexión Final (3 puntos)

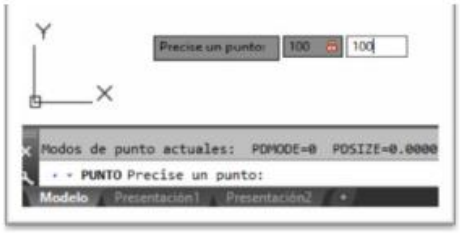
- ¿Por qué es importante seguir normas de presentación y estilos al realizar un plano técnico?
- ¿Qué ventajas ofrece AutoCAD en comparación con el dibujo manual en cuanto a precisión y edición?

Ejercicio 10. Entidades de dibujo en Autocad

Instrucciones: Realiza el Ejercicio 9 de entidades de dibujo en AutoCAD.

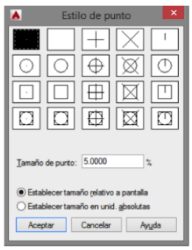
Se van a utilizar comandos de creación de objetos básicos. Todos se encuentran reunidos en la ficha "Dibujo" de la cinta "Inicio".

Inicio > Dibujo > Varios puntos> Comando Punto. El punto es la estructura más simple que maneja AutoCAD. Para dibujar un punto basta con dar clic en el lugar en donde se desea dibujar. Esto puede ser muy impreciso por lo que conviene definir sus coordenadas a través de la entrada interactiva o de la ventana de comandos.



Dibujar puntos con los diferentes métodos de entrada con las coordenadas cartesianas: (100,100), (100,200), (200,200), (200,100). Se puede cambiar el formato del punto: Inicio > Utilidades> Tipo de punto> Comando > Comando Tipo punto.

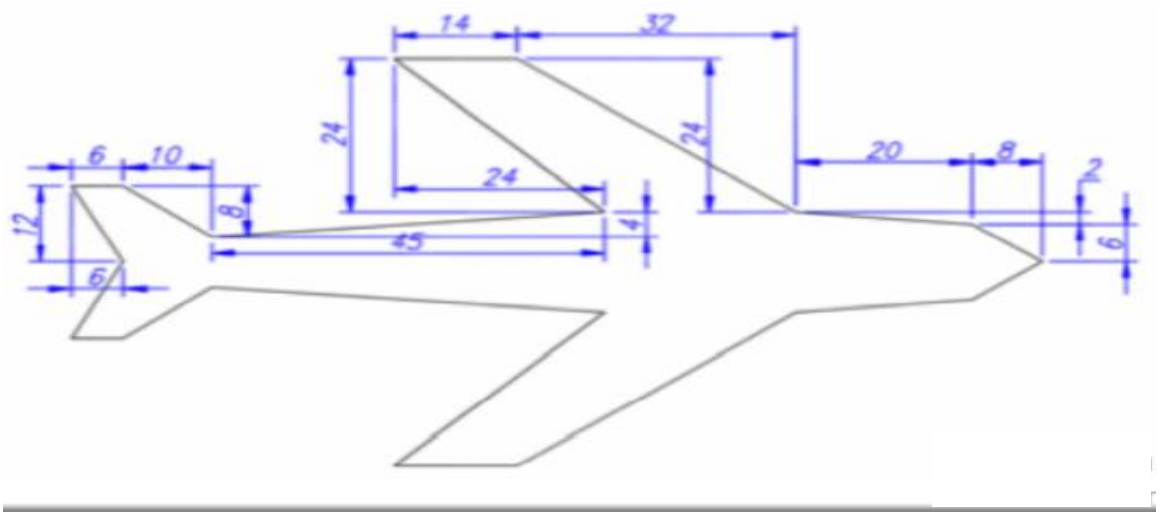
Cambiar el formato de punto a 



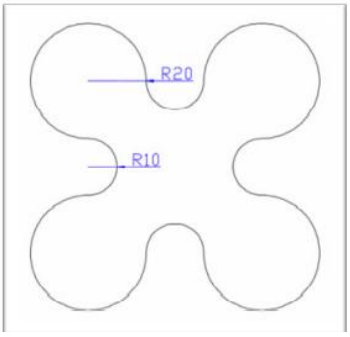
Crear un conjunto de segmentos consecutivos, determinados por un punto inicial y un punto final. El punto final de un segmento se convierte automáticamente en el inicial del siguiente. Para finalizar la creación de segmentos, pulsamos ESC. Cuando estamos dibujando un segmento tenemos disponible la opción "Deshacer", que elimina el último punto definido. Inicio > Dibujo > Línea

Para acceder a las opciones del menú contextual, pulsa sobre la flecha de cursor abajo  . A partir del tercer segmento, se tiene disponible la opción "Cerrar", que finaliza la creación de segmentos haciendo

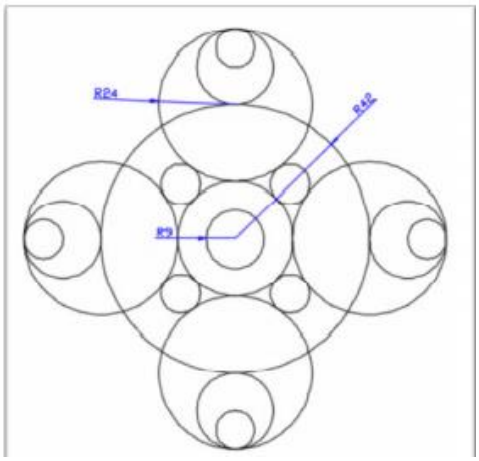
coincidir el punto final del último con el inicial del primero, creando así una figura poligonal cerrada. Realizar el siguiente dibujo utilizando el comando Línea.



Dibujar un arco de círculo, hay hasta 11 formas distintas de dibujarlo, todas combinaciones de dos puntos más otro punto, o una longitud, o un ángulo. La opción por defecto es "3 puntos", definimos el punto inicial, uno intermedio y el final, el programa dibuja el arco que pasa por los tres. Los arcos se trazan en sentido contrario a las agujas del reloj, aunque se puede invertir este funcionamiento pulsando la tecla "Ctrl". Los elementos que forman parte de las opciones son: Inicio, es el punto inicial del arco. Centro, es el centro de la circunferencia que extiende el arco. Fin, es el punto de finalización del arco. Ángulo, un ángulo negativo supone trazar el arco en sentido horario. Longitud, hace referencia a la cuerda del arco, al trazarse siempre en sentido antihorario, una longitud negativa supone un ángulo mayor a 180°. Dirección es la de la recta tangente al punto inicial. Radio, nunca podrá ser menor a la mitad de la distancia entre los puntos inicial y final. Para trazar arcos de más de 180° se marca un radio negativo. Continuo, crea un nuevo arco a partir del punto final del último objeto dibujado, con la tangente común en ese punto y definiendo un nuevo punto como finalización del arco. Realizar el siguiente dibujo (arco, referencia a objetos). Inicio > Dibujo > Arco.

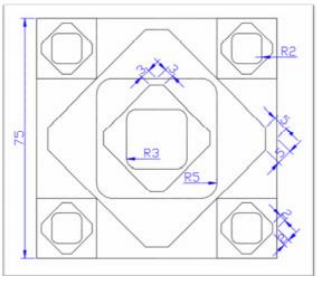


Hay varias formas de dibujar un círculo: Inicio > Dibujo > Círculo  Centro y radio, primero se define su punto central y luego el radio. Centro y diámetro, igual que el anterior, pero con el diámetro. 2 puntos, que definen su diámetro y el centro estará en el punto central del segmento que definen. 3 puntos, se ajusta el círculo al que pasa por los tres puntos. Tangente, Tangente y radio, se ajusta a dos objetos designados como tangentes y a un radio definido. Tangente, Tangente, Tangente, se ajusta a tres objetos designados como tangentes. Realizar el siguiente dibujo usando los comandos: Círculo, Referencia a objetos.



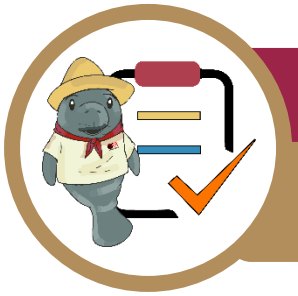
Rectángulos. Inicio > Dibujo > Rectángulo  . Para dibujar un rectángulo hay que definir dos puntos, que son esquinas opuestas. Hay opciones avanzadas en el dibujo de rectángulos. Antes de definir el primer vértice, en la ventana de comandos aparecen las opciones: Chaflán, define esquinas facetadas, con dos medidas adicionales correspondientes a las distancias desde el inicio del corte hasta el vértice. Elevación, indica a que altura sobre el plano horizontal se desea dibujar el rectángulo. empalme, define esquinas redondeadas con una medida adicional, que es el radio de redondeo. Alt-objeto, define la altura que tendrá en el eje perpendicular al plano. Grosor, establece el espesor de las líneas.

Al tener definido el primer vértice, se tienen más opciones antes de fijar el segundo: área, permite definir un rectángulo con un área determinada, hay que definir bien la longitud, bien la anchura y el programa calcula la otra dimensión. Cotas, definiendo numéricamente la anchura y la longitud, traza el rectángulo pivotando respecto del vértice fijado. rotación, establece ángulo de inclinación de la base del rectángulo antes de permitir fijar el segundo vértice. Realizar el siguiente dibujo con los comandos: Rectángulo, Forzar cursor.



Bibliografía:

https://www.autodesk.mx/



INSTRUMENTO DE EVALUACIÓN

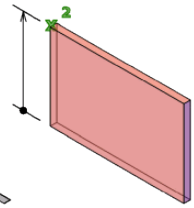
LISTA DE COTEJO 8

Ejercicio 10: Entidades de dibujo en Autocad

DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Dibujos en Autocad		**Fecha**	
Formación laboral básica: Robótica **Submódulo 2:** Control industrial		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Ingresa a Autocad.			1		
2.	Utiliza el grupo Dibujo con los comandos especificados.			2		
3.	Crea los dibujos de la forma indicada.			2.5		
4.	Utiliza la ventana de comandos para crear los dibujos.			2.5		
5	Crea las cotas en cada uno de los dibujos.			2		
				CALIFICACIÓN		

Lectura 10. Extrusión, cortes e impresiones en Autocad



Crear un sólido 3D a partir de un objeto que encierra un área o una superficie 3D a partir de objeto con extremos abiertos. Los objetos se pueden extruir de forma ortogonal desde el plano del objeto de origen, en una dirección especificada o a lo largo de una ruta seleccionada, también se puede especificar un ángulo de inclinación. La variable de sistema DELOBJ determina si los objetos de origen o la ruta seleccionada se suprimen automáticamente al crear el sólido o la superficie, o si se solicita.

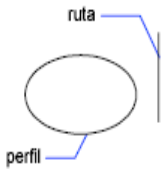
Se pueden especificar los objetos que se desean extruir. Para seleccionar subobjetos de cara y arista se mantiene presionada la tecla Ctrl mientras se seleccionan.

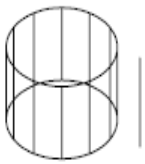
Modo. Controla si el objeto extruido es un sólido o una superficie. Las superficies se extruyen como superficies NURBS o superficies de procedimiento, según el valor de la variable de sistema SURFACEMODELINGMODE.

Altura de extrusión. Extruye los objetos seleccionados a lo largo del eje Z positivo o negativo. La dirección se basa en el SCP que estaba activo cuando se creó el objeto o para selecciones múltiples en el SCP original del último objeto creado.

Dirección. Precisa la longitud y la dirección de la extrusión con dos puntos especificados. La dirección no puede ser paralela al plano de la curva de barrido creada por la extrusión.

Trayectoria. Especifica la trayectoria de extrusión basándose en un objeto seleccionado. La trayectoria se desplaza al centro de gravedad del eprfil. A continuación, el perfil del objeto designado se extruye a lo largo de la trayectoria seleccionada para crear sólidos o superficies. La trayectoria no debe encontrarse en el mismo plano que el objeto, ni tener áreas de gran curvatura. La extrusión se inicia en el plano del objeto y mantiene la orientación relativa a la trayectoria. Si la trayectoria



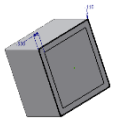


contiene segmentos que no son tangentes, el programa extruye el objeto a lo largo de cada segmento, a continuación, ingletea la unión a lo largo del plano que biseca el ángulo formado por los segmentos. Si la trayectoria está cerrada, el objeto debe encontrarse en el plano del inglete. Así se permite que coincidan las secciones inicial y final del sólido. Si el objeto no se encuentra en el plano del inglete, se gira el objeto hasta quedar situado en el mismo. Los objetos con varios bucles se extruyen de forma que todos los bucles aparezcan en el mismo plano en la sección final del sólido extruido.

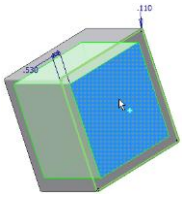
Ángulo de inclinación. Especifica el ángulo de inclinación de la extrusión. Los ángulos positivos se inclinan hacia adentro desde el objeto base. Los ángulos negativos se inclinan hacia afuera. El ángulo por defecto, 0, extruye un objeto 2D de forma perpendicular a su plano 2D. todos los objetos y bucles seleccionados se inclinan en la misma medida. La especificación de un ángulo de inclinación o de una altura de extrusión extensa puede hacer que el objeto o partes de este se inclinen hacia un punto antes de alcanzar la altura de la extrusión. Los bucles individuales de una región se extruyen siempre con la misma altura. Cuando un arco forma parte de una extrusión cónica, el ángulo de este arco permanece constante, mientras que el radio del arco si cambia. Ángulo de inclinación, especifica la inclinación entre -90 y +90 grados. Precise dos puntos, especifica el ángulo de inclinación basándose en dos puntos especificados. El ángulo de inclinación es la distancia entre los dos puntos especificados. Arrastrar el cursor horizontalmente para especificar y previsualizar el ángulo de inclinación. También se puede arrastrar el cursor para ajustar y previsualizar la altura de la extrusión. El origen de entrada dinámica debe colocarse en la forma extruida, en la proyección del punto en la forma. Al seleccionar el objeto extruido, la posición del pinzamiento de inclinación será el punto correspondiente del origen de entrada dinámica en la cara superior de la extrusión.

Expresión. Se puede escribir una fórmula o una ecuación para especificar la altura de la extrusión.

Extrusión de corte.

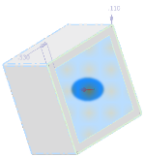


1. Usar ViewCube para cambiar la vista y hacer que el boceto esté visible.

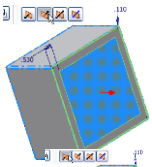


2. Pulsar el comando Extrusión  a continuación, pulsar dentro del contorno interior del boceto., la vista preliminar indica la distancia de extrusión actual.

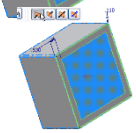
3. En el campo Extensión del cuadro de diálogo Extrusión, seleccionar Hasta-Siguiente en el menú desplegable. Si utiliza la mini barra de herramientas, seleccionar Hasta siguiente cara/cuerpo en el menú desplegable.
4. En el cuadro de diálogo Extrusión o desde la mini barra de herramientas, dar clic en  Cortar. Dentro de la región seleccionada de la pieza de la caja, se debe ver un indicador de dirección que señala a la parte interior de la caja. El indicador muestra la dirección de la extrusión de corte. Aunque aparezca atenuado el indicador de dirección se puede ver en la parte resaltada de la imagen.



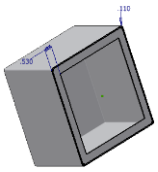
5. Para ver el indicador de un modo más claro, pulsar el botón Cambiar dirección en el cuadro de diálogo o desde la mini barra de herramientas.



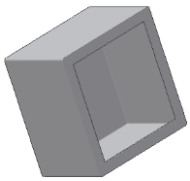
6. Pulse en el otro botón Cambiar dirección, asegurarse de que el indicador señale hacia el interior de la caja.



7. Pulse Aceptar para crear el corte. Como se ha seleccionado Hasta-Siguiente en el menú Extrusión o Hasta siguiente cara/cuerpo desde la mini barra de herramientas, el corte termina en la siguiente cara que encuentra. Que puede ser la cara posterior del cuadro.



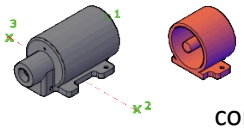
8. En el navegador dar clic con el botón derecho en Boceto 1 (anidado dentro de Extrusión 1), a continuación, eliminar la marca de verificación que aparece junto a Visibilidad.



9. Guardar la pieza.

La creación de sólidos por extrusión es el método más utilizado. El volumen extruido se genera a partir del barrido de un perfil a lo largo de un eje o una altura de extrusión. El perfil puede ser un círculo, una elipse, una polilínea 2D o 3D abierta o cerrada, una spline 2D cerrada o una región. El eje de extrusión o camino tendrá que ser una línea o polilínea 2D o 3D o incluso puede materializarse con dos puntos. Si se introduce una altura de extrusión positiva, el barrido respeta el sentido positivo del eje Z del SCP actual mientras que una altura de extrusión negativa se realiza según el sentido negativo del mismo eje. Autocad permite una visualización dinámica del sólido mediante el movimiento del ratón.

Cortes.



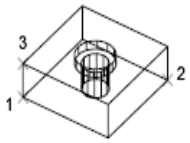
El plano de corte se define con 2 o 3 puntos, especificando un plano principal del SCP o seleccionando un objeto plano o de superficie, pero no una malla. Se pueden conservar o ambos lados de los objetos cortados. Los objetos sólidos 3D se pueden cortar con los planos y los objetos de superficie especificados. Los objetos de superficie se pueden cortar solo con los planos especificados. Las mallas no se pueden cortar ni utilizar como superficie cortante directamente.

Los objetos cortados conservan las propiedades de color y capa de los objetos originales, sin embargo, el objeto sólido o de superficie obtenido no conserva la historia de los objetos originales. Los comandos que se pueden utilizar son:

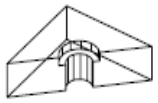
- Objetos para cortar. Especifica qué sólido 3D u objeto de superficie desea cortar. Si se selecciona una malla, se puede optar por convertirla en un sólido 3D o una superficie antes de finalizar la operación de corte. Punto inicial del plano cortante, objeto plano, superficie, eje Z, ver, XY, YZ, XZ, 3 puntos.
- Punto inicial del plano cortante. Seleccionar primero de los dos puntos que definen la orientación del plano cortante. Con esta opción el plano cortante será siempre perpendicular al plano XY del SCP actual. Cuando haya especificado el segundo punto en el plano, puede elegir si desea mantener ambos lados del objeto cortado o puede especificar otro punto del lado del plano que se desea conservar: Segundo punto del plano establece el segundo de los dos puntos del plano cortante, si el segundo punto no se encuentra en el plano XY del SCP se proyecta en el plano. Especificar un punto en el lado que se desea mantener, mantener ambos lados.
- Objeto plano. Alinea el plano de corte con un plano que contiene el círculo, la elipse, el arco circular o elíptico, la spline 2D, la polilínea 2D o la polilínea plana 3D que se haya designado. Se designa un círculo, una elipse, un arco, una spline 2D o una polilínea 2D, especificando el objeto plano que define el plano de corte. También se puede seleccionar un objeto de polilínea plana 3D. Especificar un punto en el lado que se desea mantener. Mantener ambos lados.
- Superficie. Alinea el plano de corte con la superficie seleccionada. Designar una superficie, especificar la superficie de corte. No se puede especificar una malla, una cara 3D ni objetos engrosados como superficie de corte. Seleccionar el objeto cortado en el lado que se desea mantener. Mantener ambos lados.
- Eje Z. define el plano de corte mediante la especificación de un punto en el plano y otro en el eje Z (normal) del plano. Precise un punto en el plano de sección, estableciendo un punto en el plano cortante. Precise un punto en el eje Z (normal) del plano, especificando un punto que define el eje perpendicular al plano cortante. Especificar un punto en el lado que se desee mantener. Mantener ambos lados.
- Ver. Alinea el plano de corte en paralelo con el plano de vista de la ventana gráfica actual. Al indicar un punto se determina la ubicación del plano de corte. Precise un punto en el plano de vista actual, establecer un punto del objeto para comenzar el corte. Especificar un punto en el lado que se desee mantener. Mantener ambos lados.
- XY. Alinea el plano de corte con el plano XY del SCP actual, especificar un punto para definir la ubicación del plano de corte. Punto en el plano XY, alinea el plano de corte en paralelo con el plano XY del SCP y pasando a través de un punto especificado. Especificar un punto en el lado que se desea mantener. Mantener ambos lados.
- YZ. Alinea el plano de corte con el plano XY del SCP actual. Especificar un punto para definir la ubicación del plano de corte. Punto en el plano YZ, alinea el plano de corte en paralelo con el plano YZ del SCP y pasando a través de un punto especificado. Especificar un punto en el lado que se desea mantener. Mantener ambos lados.
- XZ. Alinea el plano de corte con el plano XZ del SCP actual. Especificar un punto para definir la ubicación del plano de corte. Punto en el plano XZ, alinea el plano de corte en paralelo con el plano

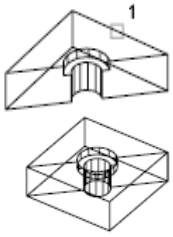
XZ del SCP y pasando a través de un punto especificado. Especificar un punto en el lado que se desea mantener. Mantener ambos lados.

- 3 puntos. Define el plano de corte mediante tres puntos.



plano de corte de 3 puntos


objeto cortado



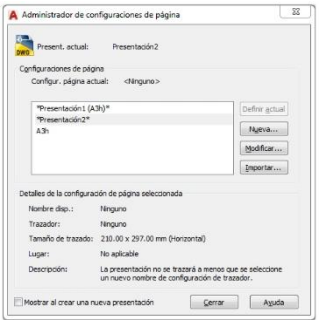
- Especificar un punto en el lado que se desea mantener. Utiliza un punto para determinar qué lado del objeto cortado se conserva. El punto no puede estar en el plano de corte.
- Mantener ambos lados. Conserva ambos lados de los objetos cortados

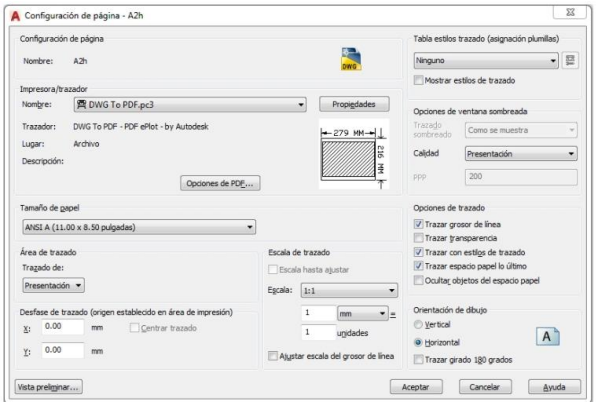
Impresiones.

La orientación del plano, la escala, los márgenes, el tipo de línea, es importante para la impresión en Autocad, para ello se necesita conocer: cómo trabajan las impresoras, como trabaja Autocad. Con esto se puede imprimir lo que se desee, donde, cuando y como se desee. No importa si cambias de computador, de plotter o de centro de trabajo. Si algo no sale bien con esto se sabe dónde buscar el problema. Como se dibuje en metros o en milímetros es calve para sacar la escala cuando imprimas. Las impresoras hablan en milímetros, es su idioma y no son bilingües.

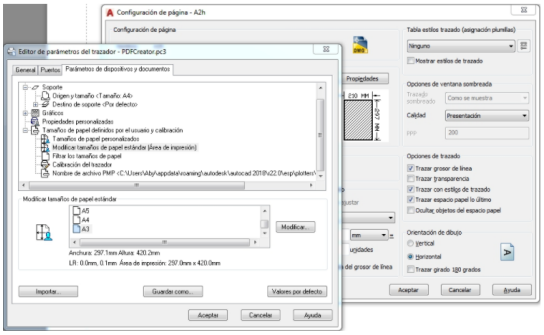
El espacio papel. En el espacio modelo puedes elegir las unidades de trabajo, pero en la presentación no. La presentación está hecha para imprimir, así que le han puesto el mismo idioma que a las impresoras en milímetros. Si se configura bien el espacio papel, imprimir será un clic, a escala 1:1 y saldrá perfecto. Saber usar el espacio papel también conocido como "Presentación" facilita mucho el trabajo a la hora de imprimir. Ayuda si se tiene que usar el espacio modelo para montar los planos, es el mismo proceso, pero a la inversa.

1. Crea una nueva configuración de página. Colocarse en la ficha de la presentación que se vaya a utilizar. Ve a Archivo > Administrador de configuración de página. En un proyecto normalmente los planos tienen casi todos, el mismo formato, pero no necesita configurar todas las presentaciones, una por una. Se puede crear una configuración de página y asignarla a todas a todas las presentaciones que se quiera, esto ahorra tiempo y trabajo. Se pueden duplicar las presentaciones y se lleva detrás la configuración de página. También se puede aprovechar para otros dibujos con la opción de importar. Configurar la página, en vez de modificar la presentación, dar clic en Nueva. Pedirá que des un nombre, se abre una ventana muy parecida a la de imprimir. Lo más importante del proceso es la impresora y el tamaño del papel.





2. Elegir la impresora. Se puede usar PDF Creator, ya viene con los tamaños de papel que se suele usar, serie DIN-A. Solo hay que modificar los márgenes y dejarlos a cero. De acuerdo a esto, se elige PDF Creator y clic en Propiedades > Pestaña Parámetros de dispositivo y documentos > Tamaños de papel definidos por el usuario y calibración > Modificar tamaños de papel estándar (área de impresión). No se va a modificar el tamaño de papel, solo se va a cambiar el área de impresión, es decir, los márgenes. Dar clic n cualquier formato, se verá justo debajo de esa ventana el tamaño y el área de impresión, ligeramente menor. Lo que se debe hacer es que el área de impresión sea exactamente el tamaño del papel, clic en Modificar y dejar todos los márgenes a cero. Siguiente > Finalizar > Aceptar. Ahora se elige el tamaño de papel que se ha modificado.



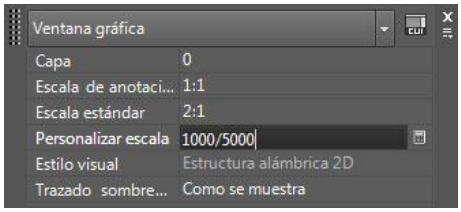
3. La escala en las ventanas gráficas. En la cinta de opciones o barra de herramientas, das clic Presentación y se crea una nueva ventana gráfica. Hay que tener cuidado con la escala de la ventana gráfica. Cuando quieres hacer un plano de situación y emplazamiento, necesitas una escala de 1:5000. Lo cual significa que una unidad en el papel son 5000 en la realidad. Al hablar de un plano de situación, el dibujo en el espacio modelo estará en metros. Pero el espacio papel y la impresora está en milímetros. El espacio modelo es la realidad, el espacio papel es el dibujo, la escala es el dibujo dividido por la realidad. E = d/r. No se puede dividir milímetros entre metros. Hay dos opciones, da igual como se haga, el resultado es el mismo:

- Pasar los metros a milímetros.
- Pasar los milímetros a metros.

Al querer un plano a escala 1:5000, en la ventana gráfica se tiene que escribir una de estas dos opciones:

- 1:5.
- 1000:5000

Se puede inmovilizar la ventana para no cambiar la escala por accidente.



4. Imprimir en Autocad. Se ha creado una configuración de página que sirve para todos los planos del mismo tamaño. Se ha dibujado tal y como se tiene costumbre. Se ha montado el plano en la Presentación correspondiente, se ha ajustado la escala de la ventana gráfica. Ahora solo hay que dar clic en imprimir. Archivo > Trazar hay que cambiar el estilo de trazado, las plumillas, vista preliminar por si acaso y Aceptar. Con esto se realiza la impresión en Autocad.

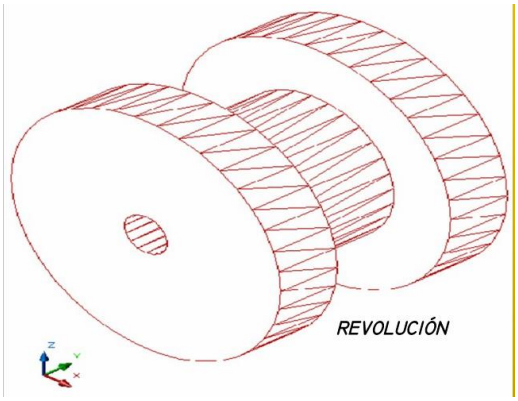
Bibliografía.

Imprimir | AutoCAD 2019 | Autodesk Knowledge Network

Actividad 10. Extrusión, cortes e impresiones en Autocad

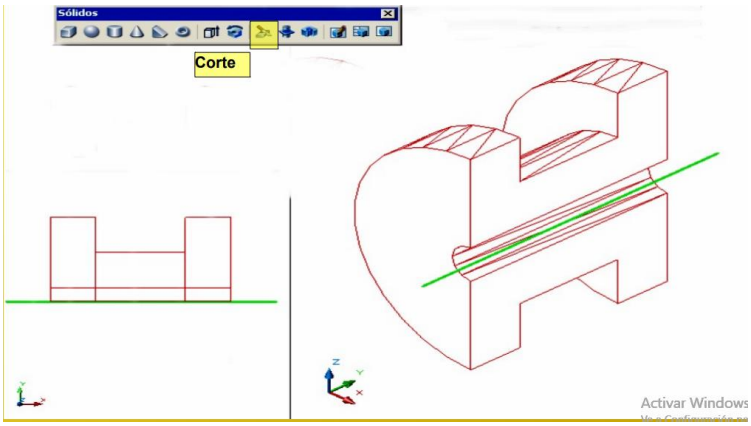
Instrucciones: Después de realizar la Lectura 10 crea el siguiente dibujo en 3D, el corte y la impresión en PDF.

Dibujo.



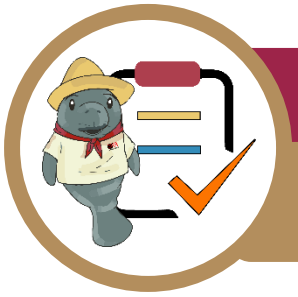
Corte.

Un sólido se puede cortar a lo largo de su intersección con un plano que se precisa. Se cortará el objeto creado a lo largo de un plano paralelo a los ejes Y y Z actuales, se conservará la mitad de la izquierda para ver el interior de la pieza.



Imprimir.

No olvides la impresión para enviarlo a un archivo PDF



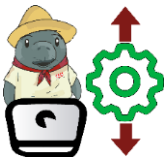
INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 9

Actividad 10: Extrusión, cortes e impresiones en Autocad

DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Dibujos en Autocad		**Fecha**	
Formación laboral básica: Robótica **Submódulo 2:** Control industrial		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Ingresa a Autocad.			1		
2.	Aplica la extrusión al crear el dibujo.			2		
3.	Realiza el corte al dibujo paralelo a los planos Y y Z.			2.5		
4.	Muestra la parte izquierda del interior de la pieza.			2.5		
5	Realiza la impresión enviando el dibujo a formato PDF.			2		
				CALIFICACIÓN		

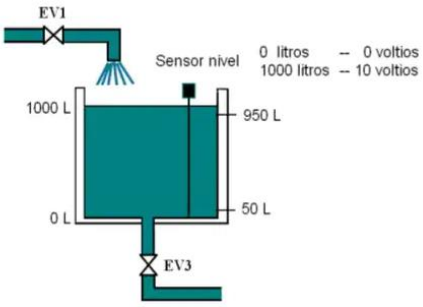


Práctica 3. PLC en Autocad

Instrucciones: En equipo de 5 integrantes realizar el diseño con el PLC en AutoCAD para el desarrollo de la situación didáctica: Robots industriales ¿en mi escuela?

Problemática planteada.

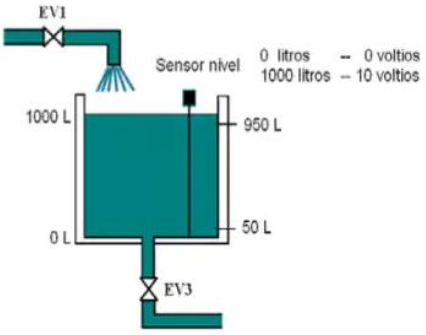
En el plantel se desea supervisar el llenado de un depósito o cisterna, de tal forma que la electroválvula EV1 se active para el llenado del depósito cuando a éste le queden tan sólo 59 litros y que se desactive cuando tenga 950 litros. Para el control se dispone de un sensor de nivel analógico calibrado para la lectura entre 0 y 1000 litros. La señal que entrega el sensor de nivel está comprendida entre 0 y 10 V para los valores mínimo y máximo respectivamente.

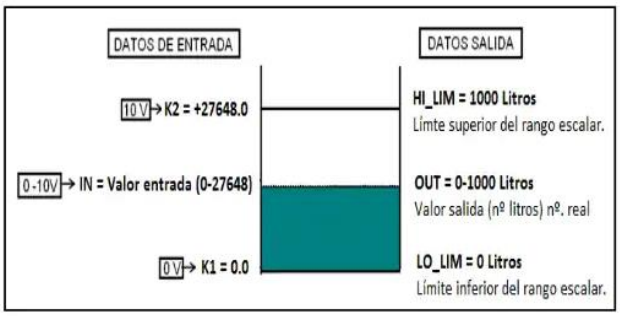


System			Voltage measuring range		Range
	dec.	hex.	1 to 5 V	0 to 10 V	
118.515 %	32767	7FFF	5.741 V	11.852 V	Overflow
117.593 %	32512	7F00			
117.589 %	32511	7EFF	5.704 V	11.759 V	Overrange
	27649	6C01			
100.000 %	27648	6C00	5 V	10 V	
75 %	20736	5100	3.00 V	7.5 V	
0.003617 %	1	1	1 V + 144.7 µV	0 V + 361.7 µV	Nom. range
0 %	0	0	1 V	0 V	
-0.003617 %	-1	FFFF	1 V - 144.7 µV	negative values not possible	Underrange
-17.593 %	-4864	ED00	0.296 V		
	-4865	ECFF			Underflow
≤ -17.596 %	-32768	8000			

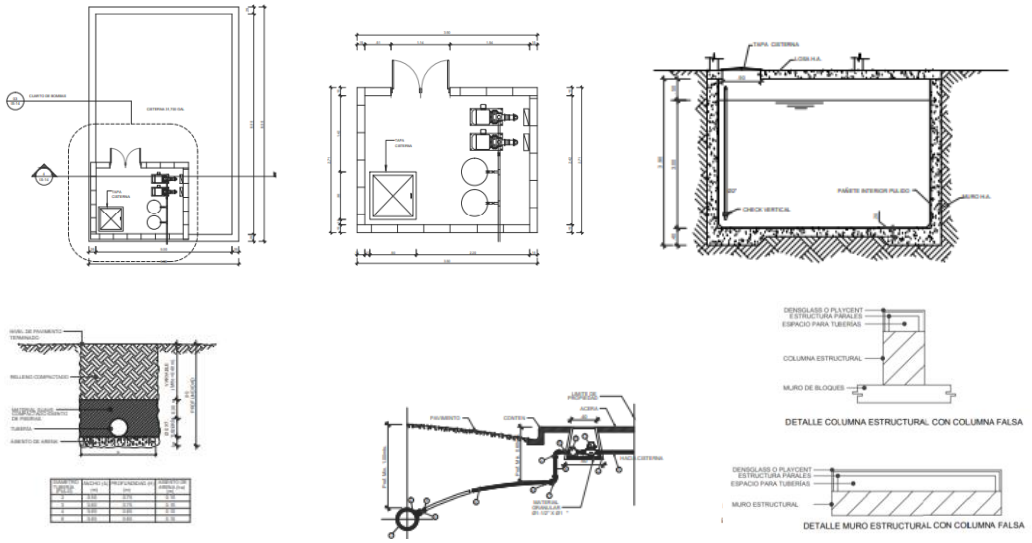
Como se observa para los valores unipolares entre 0 y 10 V de la entrada analógica, el convertidor A/D interno del PLC entrega valores enteros comprendidos entre 0 y 27648 respectivamente. El escalado de estos valores se realiza mediante la función "Escalar valores" SCALE (FC105) disponible en step 7. La función FC105 toma un valor entero en la entrada IN y lo convierte en un valor real, convirtiéndolo a escala en un rango comprendido entre un límite inferior (LO-LIM) y un límite superior (HI-LIM). El resultado de la función SCALE es un número real que se obtiene en la salida OUT.

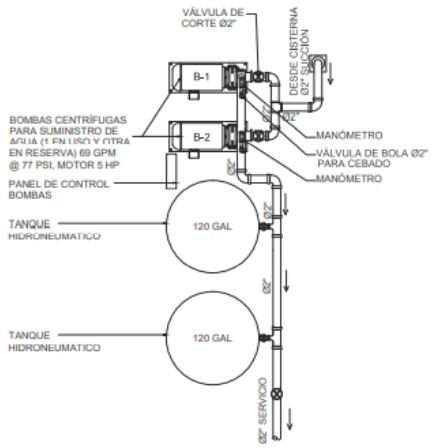
En primer lugar, se va a representar gráficamente los valores necesarios para la función FC105 y el dibujo en AutoCAD.

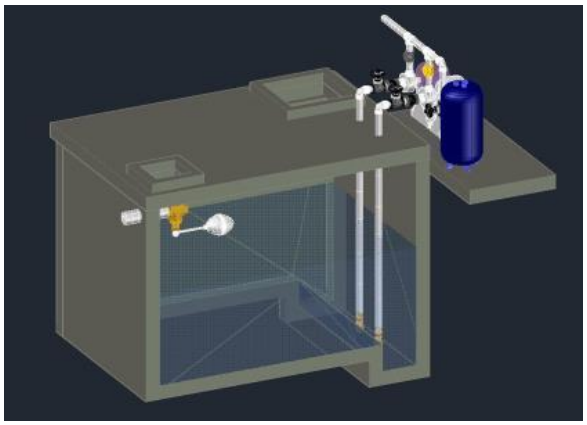




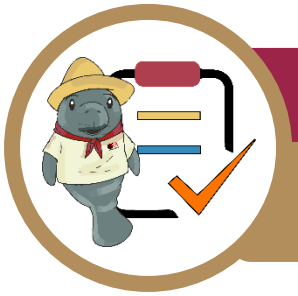
En segundo lugar, hacer el diseño completo en AutoCAD del lugar en donde esté el depósito, la ubicación del sensor y del PLC para el control que se desea realizar. Revisen el siguiente ejemplo.







En tercer lugar, imprimir en un archivo PDF.



INSTRUMENTO DE EVALUACIÓN

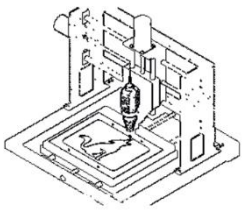
LISTA DE COTEJO 10

Práctica 3: PLC en Autocad

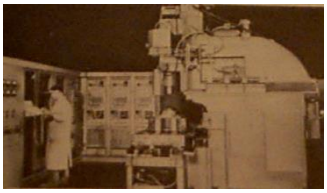
DATOS GENERALES			
Nombre(s) del alumno (s)		**Matriculas(s)**	
Producto: Reporte de práctica		**Fecha**	
Formación laboral básica: Robótica **Submódulo 2:** Control industrial		**Periodo: 2026-2027A**	
Nombre del docente:		**Firma del docente**	

No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Ingresa a Autocad.			1		
2.	Representan gráficamente la función y el dibujo.			2		
3.	Crean el diseño completo con lo especificado.			2.5		
4.	Muestran la parte izquierda de la cisterna.			2.5		
5	Realizan la impresión enviando el dibujo a formato PDF.			2		
				CALIFICACIÓN		

Lectura 11. Control numérico computarizado (CNC)



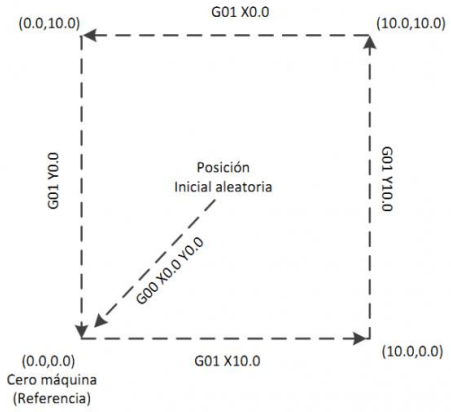
El control numérico computarizado (CNC), tuvo su origen a principios de los años 50 en el Instituto Tecnológico de Massachusets (MIT), en donde se automatizó por primera vez una gran fresadora. En esta época, las computadoras estaban en sus inicios y eran tan grandes que el espacio ocupado por la computadora era mayor que el de la máquina.



Hoy en día las computadoras son cada vez más pequeñas y económicas, debido a ello el uso del CNC se ha extendido a todo tipo de maquinaria: tornos, rectificadoras, electroerosionadoras, máquinas de coser, etc. CNC significa "Control Numérico Computarizado", en una máquina CNC a diferencia de una máquina convencional o manual, una computadora controla la posición y velocidad de los motores que accionan los ejes de la máquina. Por esto se pueden hacer movimientos que no se logran manualmente como círculos, líneas diagonales y figuras complejas tridimensionales. Las máquinas CNC son capaces de mover la herramienta al mismo tiempo en los 3 ejes para ejecutar trayectorias tridimensionales como las que se requieren para el maquinado de moldes y troqueles.

En una máquina de mecanizados en CNC, una computadora controla el movimiento de la mesa, el carro y el husillo. Una vez programada la máquina esta ejecuta todas las operaciones por sí sola, sin necesidad de que el operador esté manejándola. En el caso de una industria o taller, esto permite aprovechar mejor el tiempo del personal para que sea más productivo. El término "Control Numérico" se debe a que las órdenes dadas a la máquina son indicadas mediante códigos numéricos. Por ejemplo, para indicarle a la máquina que mueva la herramienta describiendo un cuadrado de 10 mm por lado se le darían los siguientes códigos:

G90 G71	**(cotas absolutas referidas al punto 0,0; ¨Programación en mm)**
G00 X0.0 Y0.0	(posicionamiento rápido lineal al punto 0,0 del plano XY)
G01 X10.0	(movimiento lineal de 10 mm en la dirección X positiva)
G01 Y10.0	(movimiento lineal de 10 mm en la dirección Y positiva)
G01 X0.0	(movimiento lineal de 10 mm en la dirección X negativa)
G01 Y0.0	(movimiento lineal de 10 mm en la dirección Y negativa)



Un conjunto de órdenes que siguen una secuencia lógica constituye un programa de maquinado. Dándole las órdenes o instrucciones adecuadas a la máquina, esta es capaz de maquinar una simple ranura, una cavidad irregular, la cara de una persona en autorrelieve o bajorrelieve, un grabado artístico, un molde de inyección de una cuchara o el de una botella, lo que se desee. Al principio hacer un maquinado era muy difícil y tedioso, porque se tenía que planear e indicarle manualmente a la máquina cada uno de los movimientos que tenía que hacer. Era un proceso que podía duras horas, días o semanas. Aún así, era un ahorro de tiempo comparado con los métodos convencionales.

Actualmente, muchas de las máquinas modernas trabajan con lo que se conoce como "lenguaje conversacional" en el que el programa escoge la operación que desea y la máquina le pregunta los datos que se requieren. Cada instrucción de este lenguaje conversacional puede representar decenas de códigos numéricos. Por ejemplo, el maquinado de una cavidad completa se puede hacer con una sola instrucción que especifique el largo, alto, profundidad, posición, radios de las esquinas, etc. Algunos controles incluso cuentan con graficación en pantalla y funciones de ayuda geométrica. Todo esto hace que la programación se más rápida y sencilla. También se pueden emplear sistemas CAD/CAM que generan el programa de maquinado de forma automática. Para la realización de un programa de maquinado se pueden utilizar dos métodos:

- Programación Manual. En este caso, el programa pieza se escribe únicamente por medio de razonamientos y cálculos que realiza un operario.
- Programación Automática. En este caso, los cálculos los realiza una computadora, que suministra en su salida el programa de la pieza en lenguaje máquina. Por ello recibe el nombre de programación asistida por computadora.

Programación Manual.

El lenguaje máquina comprende todo el conjunto de datos que el control necesita para la mecanización de la pieza. Al conjunto de informaciones que corresponde a una misma fase del mecanizado se le denomina bloque o secuencia que se numeran para facilitar su búsqueda. Este conjunto de informaciones es interpretado por el intérprete de órdenes. El programa de mecanizado contiene todas las instrucciones necesarias para el proceso de mecanizado. Una secuencia o bloque de programa debe contener todas las

funciones geométricas, funciones máquina y funciones tecnológicas del mecanizado, de tal forma, un bloque de programa consta de varias instrucciones.

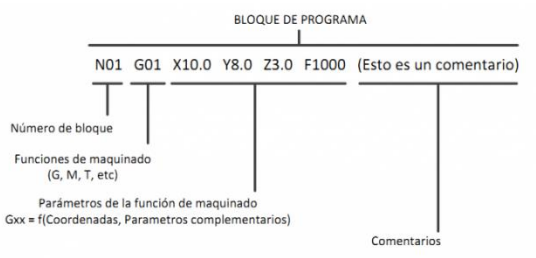
El comienzo del control numérico ha estado caracterizado por un desarrollo anárquico de los códigos de programación. Cada constructor utilizaba uno en particular. Posteriormente se vio la necesidad de normalizar los códigos de programación como condición indispensable para que un mismo programa pudiera servir para diversas máquinas que fuesen del mismo tipo. Los caracteres más usados comúnmente, regidos bajo la norma DIN 66024 y 66025, entre otros son los siguientes:

- N es la dirección correspondiente al número de bloque o secuencia. Esta dirección va seguida normalmente de un número de tres o cuatro cifras. Esta dirección va seguida normalmente de un número de tres o cuatro cifras. En el vaso del formato N03, el número máximo de bloques que pueden programarse es 1000 (N000 a N999).
- X, Y, Z son las direcciones correspondientes a las cotas según los ejes X, Y, Z de la máquina herramienta. Dichas cotas se pueden programar en forma absoluta o relativa, es decir, con respecto a la última cota.
- G es la dirección correspondiente a las funciones preparatorias. Se utilizan para informar al control de las características de las funciones de mecanizado, por ejemplo, forma de la trayectoria, tipo de corrección de herramienta, parada temporizada, ciclos automáticos, programación absoluta y relativa, etc. La función G va seguida de un número de dos cifras que permite programar hasta 100 funciones preparatorias diferentes. Ejemplos: G00 es el trayecto programado que se realiza a la máxima velocidad posible, esto es a la velocidad de desplazamiento en rápido. G01 en donde los ejes se gobiernan de tal forma que la herramienta se mueve a lo largo de una línea recta. G02 es la interpolación lineal en sentido horario. G03 es la interpolación lineal en sentido antihorario. G33 indica el ciclo automático de roscado. G77 es un ciclo automático que permite programar con un único bloque el torneado de un cilindro, etc.
- M es la dirección correspondiente a las funciones auxiliares o complementarias. Se usan para indicar a la máquina herramienta que se deben realizar operaciones tales como: parada programada, rotación del husillo a derechas o a izquierdas, cambio de útil, etc. La dirección M va seguida de un número de dos cifras que permite programar hasta 100 funciones auxiliares diferentes. Ejemplos: Moo provoca una parada incondicional del programa, detiene el husillo y la refrigeración. M02 indica el fin del programa. Se debe escribir en el último bloque del programa y posibilita la parada del control una vez ejecutadas el resto de las operaciones contenidas en el mismo bloque. M03 permite programar la rotación del husillo en sentido horario. M04 permite programar la rotación del husillo en sentido antihorario, etc.
- F es la dirección correspondiente a la velocidad de avance. Va seguida de un número de cuatro cifras que indica la velocidad de avance en mm/min.
- S es la dirección correspondiente a la velocidad de rotación del husillo principal. Se programa directamente en revoluciones por minuto, usando cuatro dígitos.

- I, J, K son direcciones utilizadas para programar arcos de circunferencia. Cuando la interpolación se realiza en el plano X-Y, se utilizan las direcciones I y J. Análogamente, en el plano X-Z se utilizan las direcciones I y K, y en el plano Y-Z las direcciones J y K.
- T es la dirección correspondiente al número de herramienta. Va seguido de un número de cuatro cifras en el cual los dos primeros indican el número de herramienta y los dos últimos el número de corrección de estas.

Estructura de bloque.

Es el modo de dar órdenes a la máquina para que las ejecute. Esto tiene ciertas características que se deben cumplir. La máquina ejecuta las órdenes (operaciones) de acuerdo con los datos entregados por dicha operación, por lo que cada orden tiene una estructura definida. A cada orden se le denomina block o b loque de programa. De forma general cada bloque tiene la siguiente estructura:



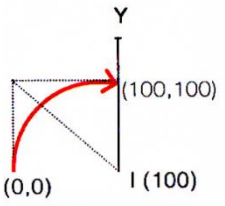
a) Número de bloque (número de línea de programa).
b) Código de orden de configuración (función de maquinado).
c) Parámetros de la función de maquinado (Coordenadas X, Y, Z y parámetros complementarios).
d) Comentarios.

Formato de bloque.

El modo básico de comunicarse con la máquina herramienta es a través de los elementos que forman la estructura de un bloque de instrucciones, en donde cada uno de los caracteres alfanuméricos tienen un significado y una representación propia.

Comandos G (funciones de desplazamiento de la fresa).

Los comandos G son las órdenes más utilizadas. Son las órdenes de movimientos de las herramientas. Son las funciones básicas del lenguaje de programación G y las que determinarán las coordenadas y la forma final de la pieza mecanizada. Ejemplo: G2: Arco de circunferencia. Datos requeridos: Coordenada de punto final, radio de la curva. G2 X100 Y100 I100 F1000.



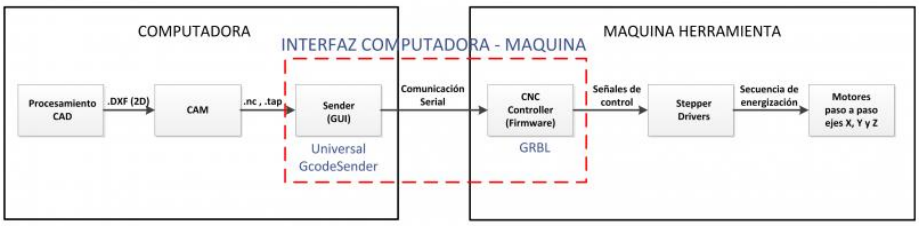
El proceso para transformar un modelo virtual de 2 o 3 dimensiones en una pieza de maquinado final pasa por seis etapas, las cuales son realizadas de forma individual o en etapas agrupadas, dependiendo de cada fabricante o diseñador de máquinas CNC, tanto en hardware como en software. Estas etapas son:

- Diseño asistido por computadora (CAD).
- Manufactura asistida por computadora (CAM).
- Envío de código G hacia la máquina (Sender)
- Controlador CNC.

- Drivers de motores paso a paso (Stepper Driver).
- Movimiento de ejes (Motores paso a paso).

Las 3 primeras etapas se ejecutan en la computadora y las siguientes 3 etapas son parte de la máquina CNC.

Hoy día los equipos CNC con la ayuda de los lenguajes conversacionales y los sistemas CAD/CAM permiten a los usuarios producir piezas con mucha mayor rapidez y calidad, sin necesidad de tener una alta especialización. El código G es un formato de texto que se pude escribir con la mano generada por un script. Aplicaciones de CAM se utilizan generalmente para generar el código G. Se utiliza principalmente las extensiones de archivo .tap y .nc, aunque existen decenas de extensiones que cumplen la misma función. Finalmente, todas son extensiones de texto tipo .txt. Se puede utilizar cualquier editor de texto para hacer o editar el archivo.



Un controlador CNC que permite la calibración, configuración y el control de fresadoras CNC. Es el encargado de procesar cada línea de proceso de un programa de maquinado (programa en código G) para que la máquina herramienta ejecute los movimientos necesarios para realizar una rutina de fresado.

Un software CNC Controler ejecuta cuatro unidades principales de procesamiento de control numérico:

- Unidad de entrada y salida de datos. Sirve para introducir los programas de mecanizado en el equipo de control numérico, utilizando un lenguaje inteligible para éste.
- Unidad de memoria interna e interpretación de órdenes. Tanto en los equipos de programación manual como en los de programación mixta, la unidad de memoria interna almacenada no sólo el programa sino también los datos máquina y las compensaciones (aceleración y desaceleración, compensaciones y correcciones de la herramienta, etc.). Son los llamados datos de puesta en operación.
- Unidad de cálculo. Una vez interpretado un bloque de información, esta unidad se encarga de crear el conjunto de órdenes que serán utilizadas para gobernar la máquina herramienta.
- Unidad de enlace con la máquina herramienta. La función principal de un control numérico es gobernar los motores (servomotores) de una máquina herramienta, los cuales provocan un desplazamiento relativo entre el útil y la pieza situada sobre la mesa. Si consideramos un desplazamiento en el plano, será necesario accionar dos motores en el espacio, tres motores y así sucesivamente.

El control numérico involucra diferentes áreas de conocimiento que son necesarias para el mejor aprovechamiento de la tecnología disponible, dichos conocimientos están íntimamente relacionados y se vuelve imperiosa la necesidad de manejarlos de forma simultánea.

Sender. Es un software que tiene como misión principal enviar la información de código G a través de un protocolo de comunicación desde la computadora hacia la máquina también controla el flujo de datos enviados funcionando como "Buffer", de modo que, si una máquina CNC acumula muchas instrucciones o queda vacía de datos, puede comunicarse con el programa Sender para detener el envío de datos o para solicitar más.

Bibliografía.

http://repositorio.utp.edu.co/dspace/bitstream/handle/11059/7012/6219023P649.pdf?sequence=1

http://www.gravotech.com.mx/productos-y-servicios/equipos-y-accesorios-de-grabado/grabado-marcaje-y-corte-laser

DISEÑO, CONSTRUCCION Y PROGRAMACION DE UN PROTOTIPO DE MAQUINA CNC, PARA EL FRESADO Y PERFORADO D (core.ac.uk)

Actividad 11. Control numérico computarizado (CNC)

Instrucciones: Después de realizar la Lectura 11 analiza los siguientes códigos G, escribe lo que corresponde a cada parte de dichos códigos.

Código.

11 G1 F900 X197.600 Y29.900 E19.82400

Respuesta.

- 11. Es el número de línea del código y se usa como referencia. Es la línea 11 del programa que se está ejecutando.
- G/M. en color azul indica que es un comando de tipo indicado por la letra. El código G1 corresponde al comando: Movimiento coordinado a velocidad de avance.
- F. Indica la velocidad en este caso 900.
- X/Y/Z. son las coordenadas de posición.
- E. Movimiento del alimentador.

Código.

G01 X1 Y1 F20 T01 M03 S500

Respuesta.

- G01. Realiza un movimiento de alimentación lineal.
- X1/Y1. Mover a las coordenadas indicadas en X e Y.
- F20. Mover a una velocidad de avance de 20.
- T01. Utilizar la herramienta 1 para hacer el trabajo.
- M03. Enciende el husillo.
- S500. Establece una velocidad de husillo de 500.

Se puede ver la simulación en el software: www.fagorautomation.com **o solicítalo al docente.**

Situación didáctica 2
¿Mi escuela, con robots industriales?

Instrucciones: Con base en la Situación Didáctica planteada al principio del submódulo, diseña y construye un Robot industrial básico aplicando los conocimientos aprendidos durante el desarrollo de dicho submódulo.

Propósito.

Diseña y construye un Robot industrial básico en equipo de 5 integrantes que permite aplicar los conocimientos adquiridos durante el estudio del submódulo para presentándolo en el aula y fuera de ella, para dar a conocer la Capacitación de Robótica.

Conocimientos.

- Controladores lógicos programables (PLC).
  - Estructura interna.
  - Clasificación.
  - Puertos de entrada y salida.
  - Manejo, instalación y conexiones.
- Programación básica lenguaje escalera.
  - Ejecución.
  - Contadores
  - Temporizadores.
- Simulador de aplicación industrial.
- Circuitos de acoplamiento para alta potencia.
- Dibujo industrial.
  - Diseño asistido por computadora.
  - Vistas.

- Croquis.
- Líneas.
- Rectángulos.
- Círculos.
- Acotaciones.
- Extrusión 3D y cortes.
- Impresiones y uso del software.

❖ Control numérico computarizado (CNC).

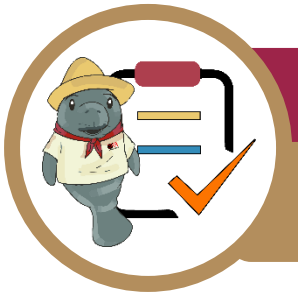
Procedimiento.

❖ Utilizar el diseño realizado en la Práctica 3. PLC en AutoCAD.

❖ Realizar el diagrama escalera del PLC para el control del depósito de agua.

❖ Simular el diagrama escalera en el software MiPlc.

❖ Lo que los estudiantes consideren necesario para el desarrollo de la situación didáctica.



INSTRUMENTO DE EVALUACIÓN

LISTA DE COTEJO 11

Situación didáctica 2: ¿Mi escuela, con robots industriales?

DATOS GENERALES						
Nombre(s) del alumno (s)				**Matriculas(s)**		
Producto: Prototipo				**Fecha**		
Formación laboral básica: Robótica **Submódulo 2:** Control industrial				**Periodo: 2026-2027A**		
Nombre del docente:				**Firma del docente**		
No.	INDICADORES	VALOR OBTENIDO		PONDERACIÓN	CAL	OBSERVACIONES Y/O SUGERENCIAS
		SI	NO			
1.	Presentan el diseño completo con extrusión realizado en AutoCAD.			2		
2.	Muestran el corte con la parte izquierda de la cisterna.			2		
3.	Muestran la impresión del dibujo en formato PDF.			2		
4.	Crean el diagrama escalera del PLC en papel.			2		
5	Crean y simulan el diagrama escalera en el software MiPlc.			2		
				CALIFICACIÓN		

Referencias

SUBMÓDULO 1

Arquitectura de microcontroladores. Extraído el 30 de junio del 2021 desde: http://www.itq.edu.mx/carreras/IngElectronica/archivos_contenido/Apuntes%20de%20materias/ETD1022_Microcontroladores/1_Arquitectura.pdf

Cómo crear, abrir o guardar un Sketch. Extraído el 6 de mayo del 2025 desde: https://docs.sunfounder.com/projects/3in1-kit-v2/es/latest/arduino_start/create_save.html

How to upload a sketch with the Arduino IDE 2. Extraído el 6 de mayo del 2025 desde: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started/ide-v2-uploading-a-sketch/

Introducción y Arquitectura de microcontroladores. Extraído el 30 de junio del 2021 desde: https://microcontroladoressesv.wordpress.com/arquitectura-de-los-microcontroladores/

Módulo de microprocesadores & microcontroladores. Extraído el 27 de junio del 2021 desde: https://repository.unad.edu.co/bitstream/handle/10596/6933/M_309696_Microp%20&%20Microc_Ing%20Elect?sequence=1

Moreno Muñoz, A. & Córcoles Córcoles, S. (2018). Arduino: curso práctico: (ed.). RA-MA Editorial.

Peña Millahual, C. A. (2017). Arduino de cero a experto: Proyectos prácticos - Electrónica, hardware y programación. Ciudad Autónoma de Buenos Aires: RedUsers. ISBN 978-987-46518-7-7.

Porcuna López, P. (2015). Robótica y domótica básica con Arduino: (ed.). RA-MA Editorial.

Qué son los microprocesadores y los microcontroladores. Extraído el 27 de junio del 2021 desde: https://techlandia.com/son-microprocesadores-microcontroladores-info_243613/

Schmidt, D. R. (2022). Arduino. Curso completo. 2ª edición. Ra-Ma Editorial.

Anexos

HIMNO DEL COBATAB

¡Oh!, Colegio de Bachilleres, impetuosa y querida
institución casa fiel del conocimiento,
hoy te canto este himno con amor,
eres rayo de esperanza del mañana,
eres la voz de la verdad.

¡Oh!, Colegio de Bachilleres, eres
luz en medio de la oscuridad.
//Colegio de bachilleres conducta
clara y firme decisión
Colegio de bachilleres tu misión
para siempre es ser mejor//

En Tabasco se ha sembrado
la semilla que un día germinara,
el impulso de la vida modernista
en progreso de toda la sociedad,
es tu memorable historia gran
orgullo para toda la región,
educación que genera cambio,
ejemplo digno en cada generación.

PORRA INSTITUCIONAL

¡Somos!
¡Somos!

Jóvenes Bachilleres
Jóvenes Bachilleres

Con Valor y Lealtad
De Norte a Sur
De Este a Oeste

Somos líderes Bachilleres del Sureste
COBATAB Unido, COBATAB Fortalecido.
Este encuentro lo gano porque lo gano
Como dijo el Peje, me canso ganso.

¡Somos!
¡Somos!

Jóvenes Bachilleres
Jóvenes Bachilleres

¡Somos!
¡Somos!

Jóvenes Bachilleres
Jóvenes Bachilleres

COBATAB Unido, COBATAB Fortalecido

COBACHITO

Colegio de Bachilleres,
Está de fiesta señores
Pues todos sus estudiantes
Hoy celebran con honores.

Que ya llegó la alegría
Es hora de motivar
Bailemos con algarabía
Cobachito nos guiará.

Allá por el acahual
En los ríos de Tabasco
Aconchado en unas ramas
O nadando sin parar.

Un manatí se ha ganado
El cariño de la gente
Cobachito le han llamado
Y no para de bailar.

Cobachito, con él vamos a ganar
Cobachito, eres espectacular
Cobachito, respetamos tu hábitat
Cobachito, mascota del COBATAB.

Mientras la orquesta se escucha
Y la porra se emociona
Los jóvenes bachilleres
A una voz ovacionan

Con orgullo representan
A una gran institución
COBATAB está presente
Y Cobachito ya llegó.