Instituto Tecnológico y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática
Maestría en Electrónica Industrial

Diseño e implementación del curso de Fundamentos de Microprocesadores y Microcontroladores

---

TESIS que para obtener el **GRADO** de
MAESTRO EN ELECTRÓNICA INDUSTRIAL

Presenti: **MARTÍN ALONSO SINSEL DUARTE**

Director **OMAR HUMBERTO LONGORIA GANDARA**

Tlaquepaque, Jalisco. 25 de febrero de 2025.

Resumen

Esta tesis presenta el proceso de diseño e implementación del curso de Fundamentos de Microprocesadores y Microcontroladores que ofrece el Instituto Tecnológico y de Estudios Superiores de Occidente a partir de agosto de 2023. Este curso atiende a los procesos de mejora continua que requiere el Consejo de Acreditación de Enseñanza de la Ingeniería (CACEI), y se ofrece como curso obligatorio para las carreras de Ingeniería en Electrónica, Ingeniería en Sistemas Digitales Embebidos e Ingeniería en Mecatrónica.

El proceso implicó una revisión del uso de los microcontroladores en la academia, a nivel licenciatura, y su vínculo con la industria. Así mismo se analizaron algunas de las estrategias didácticas utilizadas para enseñar estas tecnologías a nivel universitario.

A lo largo del documento, se desglosa la propuesta del curso en los elementos constitutivos de la guía de aprendizaje, y se presentan los materiales didácticos elaborados, programas de ejemplo y las actividades de aprendizaje, orientados a lograr los objetivos del curso.

Contenido

Introducción

El desarrollo del curso de Fundamentos de Microprocesadores y Microcontroladores enfrenta el reto de resolver varias problemáticas. Por un lado, al ser el primer curso de microcontroladores, se busca sentar las bases teórico-prácticas en las competencias de los estudiantes. Es importante que el microcontrolador seleccionado sea lo suficientemente simple para centrarse en los conceptos, para ubicar que pueden usarse en aplicaciones con relativamente pocas capacidades y terminales (pines). Por otro lado, debe de generarse la capacidad de extrapolar el conocimiento a otros microcontroladores.

Que la programación sea en lenguaje C, provoca que el estudiante ajuste el paradigma de programación en lenguaje C que muy probablemente lo aprendió para usarse en una computadora, pero sin considerar factores de cantidad de memoria y tiempo de ejecución; cuestión muy importante en sistemas embebidos.

Trabajar la programación del microcontrolador a nivel de registros (*bare metal*), sin el uso de librerías, pretende que el estudiante tenga al manual del microcontrolador como principal referencia. Lo anterior en lugar de usar librerías de las cuales no se conoce la eficiencia de la implementación y no genera, en el primer momento, conocimientos básicos de microcontroladores.

La evolución tan acelerada en los microcontroladores conlleva, desde un curso universitario inicial, el reto o problemática de fijar las bases, para que en cursos posteriores o en la vida profesional, los alumnos y egresados tengan la capacidad de utilizar tecnologías más recientes y avanzadas.

El Capítulo 1 muestra un panorama del uso de microcontroladores en el desempeño profesional de varias ingenierías. La forma en que han ido evolucionando los microcontroladores y como ha cambiado el panorama de uso. También se describe el uso de los microcontroladores en algunas ingenierías para el desarrollo de sistemas embebidos.

En el Capítulo 2 se presenta información de cómo se han constituido los cursos de microcontroladores en casi 40 años en el ITESO y en otras universidades. La cantidad de cursos de microcontroladores en diferentes programas y que tecnologías se han utilizado. De igual manera

se menciona la relación de los cursos de microcontroladores con otros cursos en diferentes Ingenierías.

El Capítulo 3 presenta las generalidades del curso propuesto: objetivos, contexto curricular, metodología, criterios de evaluación, calendario propuesto y bibliografía sugerida.

Para el Capítulo 4 se describe el detalle de cada unidad de aprendizaje. Mas allá del material didáctico presentado en el Apéndice D, lo que se busca en esta sección es complementar los objetivos de cada uno de los elementos de la unidad de aprendizaje.

En el Capítulo 5 se presentan los resultados en el curso piloto de agosto a diciembre 2023, integrando la perspectiva de los estudiantes y del profesor. Se incluye un apartado con la valoración de un profesor invitado al curso Primavera 2024.

Los Apéndices aportan información complementaria con detalles del curso.

1. Antecedentes y Contexto

1.1. Evolución de los microcontroladores.

La siguiente narrativa respecto a la evolución de los microcontroladores representa de manera muy compacta la sucesión de eventos y tecnologías desde la perspectiva del autor (Apéndice A: Experiencia del Autor en la Academia y la Industria), por lo que, si hay un interés histórico en particular, se invita al lector a la consulta de otras fuentes bibliográficas que detallan los hechos y anécdotas que han marcado la historia de la tecnología de los microcontroladores y microprocesadores.

A fines de 1971, Intel anuncia el primer microprocesador, el 4004. En los siguientes años se introdujeron una versión "gama baja" el 4004 y el 4040, ambos de cuatro bits. Además del 8008 y 8080 de 8 bits [1]. Los microprocesadores de cuatro bits fueron usados principalmente para aplicaciones donde se substituía electrónica digital con dispositivos de baja escala de integración (*LSI: Low Scale Integration*) y/o mediana escale de integración (MSI: Medium Scale Integration).

Pocos años después, se integró memoria de código, memoria de datos y dispositivos de Entrada/Salida al microprocesador y se desarrolló el microcontrolador TMS1000 de Texas Instruments fue introducida al mercado en 1974. El dispositivo incluía una CPU de 4 bits, RAM, ROM y circuitería de Entrada/Salida (E/S) todo en un solo chip.

Los dispositivos 8048/8748 fueron los primeros microcontroladores de Intel ofertados en el mercado. Era común verlos como controladores de los teclados en las computadoras de los años 80s y principios de los años 90s.

Como una mejora a la familia 8x48, Intel presenta al mercado la familia 8x51, compatible en varios aspectos tanto en los módulos internos: memoria RAM, memoria ROM, *timers* y manejo de interrupciones, como en la arquitectura del conjunto de instrucciones (ISA).

Además de Texas Instruments e Intel, muchos otros fabricantes entraron al mercado de los microcontroladores. Una de las familias que crecieron en ventas y diversidad de dispositivos fue la familia HC05 de Motorola (68HC05).

Después del Intel 8048, salió al mercado la familia 8051, un microcontrolador muy utilizado y con gran éxito, gracias a que fue fabricado después por muchos fabricantes y en una gran variedad de configuraciones. Dicho microcontrolador llegó a presentarse en aplicaciones militares y fue de los primeros en correr a 100 MHz.

El excelente conjunto de características de las subfamilias de los microcontroladores HC05 los fueron posicionando en los primeros lugares de ventas en varios segmentos de mercado, incluyendo el automotriz, segmento referente en la comparación de microcontroladores.

En los años 90s se presenta la evolución de los microcontroladores de 8 bits a los microcontroladores de 16 bits: HC11/HC12/HC16 (Motorola) y 8096 (Intel). Otros fabricantes inician a mejorar su posicionamiento en el mercado de microcontroladores de 8 bits e inician la incursión en las variantes de 16 bits: Microchip, Atmel, National Instruments, Analog Devices, Fujitsu, Infineon, Maxim, Mitsubishi y Hitachi, los últimos dos integrados en una empresa conjunta para formar Renesas.

Con características especiales, se incorporan con mayor fuerza, en los 90's, los DSP. Versiones con memorias de programa y datos internas, además de periféricos integrados. En el campo los DSPs, Texas Instruments, Motorola y Analog Devices se convierten en los líderes para los diferentes segmentos de mercado de los DSPs.

1.2. Panorama del uso de microcontroladores.

En la actualidad la gran variedad de microcontroladores nos lleva a la situación donde por cada microprocesador en uso hay una gran cantidad de microcontroladores con una gran variedad de características, desde componentes con 8 pines con costos menores a los 30 centavos de dólar hasta microcontroladores con gran número de núcleos y encapsulados de cientos de terminales.

Las aplicaciones de los microcontroladores se cuentan en una gran variedad de sistemas electrónicos de la vida cotidiana:

Sistemas médicos: tanto de uso generalizado como de uso exclusivo por especialistas. Entre los altamente comercializados encontramos pulso oxímetros, medidores de presión arterial, termómetros, medidores de azúcar en sangre. En la lista de sistemas de uso especializado encontramos: marcapasos, desfibriladores, neuro estimuladores, bombas de infusión, respiradores,

prótesis, dispositivos para rehabilitación física, sistemas de monitoreo (ECG, EEG, entre otros), equipos de laboratorio químico (centrifugadores, incubadoras, autoclaves), etc.

Sistemas automotrices: dirección del vehículo, transmisión, sistemas de luces, sistemas asociados al motor, sistemas de seguros y elevadores de vidrios, sistemas de seguridad antirrobo, sistemas de seguridad funcional (safety), sistemas de ciberseguridad, sistemas de bombas electrónicas (gasolina, aceite, enfriamiento del motor), aire acondicionado y calefacción, entretenimiento y asistencia al manejo para el conductor.

En el segmento de aplicaciones para el hogar: refrigeradores, hornos microondas, hornos eléctricos, licuadoras, estufas, lavadoras, secadoras, calentadores de agua, televisiones, reproductores de música y video.

Muchos y muy variados sectores utilizan microcontroladores de diversas complejidades, a tal punto que desde hace varias décadas se maneja el término de sistema embebido, donde comúnmente un microcontrolador se encarga de muchas de las tareas. El sistema embebido planteado como un sistema electrónico que realiza tareas específicas.

Los sistemas embebidos se enfocan en la aplicación, los principios de la tecnología computacional, el software y el hardware se pueden ajustar con la demanda estricta de la aplicación en cuanto a la funcionalidad, confiabilidad, costo, tamaño y consumo de energía. En [2] se presentan los elementos tanto de hardware como de software para desarrollar una solución con un microcontrolador de 32 bits, considerando interfases complejas típicas en este tipo de microcontroladores.

En 2004, el IEEE y la ACM desarrollaron y documentaron en [2], un currículo para sistemas embebidos donde presentan los elementos indispensables, entre ellos el microprocesador embebido, el diseño de software embebido, el sistema de interfaz para interfaces y señal mezclada, por mencionar algunos elementos, que deben ser considerados en el desarrollo del curso de Fundamentos de Microprocesadores y Microcontroladores.

En [3] se presenta un comparativo sobre la enseñanza de sistemas embebidos en una universidad de China y otra de Suecia. Donde es importante resaltar la importancia que se le da a la parte experimental. El curso de Fundamentos de Microprocesadores y Microcontroladores se considera una alta componente de trabajo experimental.

1.3. Las diferentes ingenierías usando microcontroladores.

En un gran número de profesiones se utilizan sistemas especializados basados en microcontroladores. A continuación, se muestran algunas aplicaciones de microcontroladores:

Figura 1 Motor para implantes NSK Surgic Pro. [4]

En la Figura 1, el fabricante de equipos dentales describe en [4] su equipo como "Todas las funciones para implantes NSK Surgic Pro". El dispositivo es una herramienta para realizar implantes dentales. Las funciones del manejo del teclado, pantalla, control de la velocidad del motor y otras funciones pueden ser realizadas por un microcontrolador.

La Figura 2 presenta un equipo de topografía con batería de extralarga vida. Con protección contra chorros de agua, bajas temperaturas y efectos del polvo [5].

Figura 2 Estación total Zoom 25 Pro A5. [5]

Figura 3 HOBO estación de registro de datos. [6]

La Figura 3 presenta el Sistema HOBO (Honest Observer By Onset) cuya funcionalidad se describe en mayor detalle en [6], para monitorear:

- Conductividad Eléctrica
- CO2, Oxígeno disuelto, evapotranspiración
- Humedad en hoja, humedad relativa, humedad del suelo
- Intensidad de luz
- pH
- Radiación solar
- Temperatura
- Flujo de agua
- Calidad del Agua
- Temperatura del agua
- Clima general
- Velocidad del viento

La Figura 4 muestra diversos dispositivos electrónicos: un contador de billetes, una terminal inalámbrica para realizar cargos a tarjetas electrónicas de débito o crédito y un scanner para diversas funciones.

Figura 4 Diversos equipos electrónicos para negocios

2. Definición del Problema

2.1. Establecimiento del problema

En la actualidad la metodología de aprendizaje para el uso e implementación de soluciones con microcontroladores está encaminada al desarrollo de sistemas embebidos y representa varios retos y desafíos para las universidades que imparten este tipo de cursos en el campo de las ingenierías afines a la ingeniería eléctrica y sistemas computacionales.

2.2. Objetivo General

Actualizar el curso de Fundamentos de Microcontroladores y Microprocesadores a partir del análisis y la consideración de diversos elementos antecedentes, así como el estado actual de la tecnología.

2.3. Objetivos Específicos

Analizar elementos que nutran las necesidades del curso: variantes del mismo curso a través del tiempo, cursos similares en otras universidades, cursos parecidos para varias carreras que lo demandan, atributos de egreso solicitados por CACEI (Consejo de Acreditación de la Enseñanza de la Ingeniería) y escenarios laborales profesionales donde se requieran estas competencias.

Elaborar la guía de aprendizaje como primer nivel de especificación del curso.

Diseñar y desarrollar las actividades de aprendizaje que desarrollará el estudiante: ejercicios, prácticas, proyecto final y exámenes.

Elaborar el material didáctico y programas de referencia para el microcontrolador.

2.4. Alcance

Las contribuciones de este Trabajo de Obtención de Grado (TOG) son:

i) Análisis de los elementos necesarios para poder generar la especificación del curso.
ii) Elaboración de la guía de aprendizaje.
iii) Descripción de las actividades que trabajarán los estudiantes.
iv) Elaboración de los materiales didácticos para el profesor.
v) Desarrollo de programas base para el microcontrolador que sirvan de referencia al profesor y estudiantes.

2.5. Antecedentes de cursos de microcontroladores a nivel licenciatura en el ITESO

En esta sección se describe la metodología seguida para la implementación del curso de fundamentos de microcontroladores. Primeramente, se parte de los antecedentes de los cursos previamente desarrollados en el ITESO ligados al tema de microcontroladores.

Solo como referencia, en el Apéndice A se presenta un resumen de los antecedentes profesionales del autor, tanto en la experiencia académica como industrial. Lo cual ayuda a entender el contexto del autor desde donde se genera la propuesta del nuevo curso.

La primera referencia, de la cual el autor tiene conocimiento, es el curso de Electrónica 4 (Programación e Interconexión de Microprocesadores), ofertada a los estudiantes de Ingeniería en Electrónica entre 1985 y 1989. El curso antecedente, Electrónica 2 (Sistemas Digitales), proporcionaba los elementos teóricos y prácticos referentes a sistemas digitales combinacionales y secuenciales. El curso siguiente, Electrónica 6 (Microprocesadores avanzados), estudiaba microprocesadores de 16 bits, microcontroladores y otras tecnologías digitales. Dicho curso utilizaba el microprocesador Z-80 de 8 bits. Las competencias para desarrollar eran: la programación del microprocesador en lenguaje ensamblador y la interconexión de elementos externos al microprocesador.

En términos de programación se cubrían los siguientes temas: instrucciones aritméticas y lógicas, instrucciones de transferencia con memoria, operaciones en el stack, subrutinas y manejo de interrupciones.

En términos de interconexión de elementos externos al microprocesador, se analizaban y diseñaban: circuito de reloj, circuito de *reset*, *glue logic*, memoria de programa externa (UVEPROM), memoria de datos externa (RAM), puertos E/S y sistemas para el manejo de

interrupciones. Dicho curso se mantuvo hasta mediados de los años 90, donde se cambió el microprocesador por el 8051. Se mantuvieron los temas y enfoques del curso previo. Se agregó el manejo de *timers* y elementos básicos del puerto serial.

Uno de los elementos importantes por lo que se mantuvo el 8051 por tantos años es por ser uno de los pocos microprocesadores de 8 bits, con una arquitectura usada por muchos fabricantes y con variantes de uso significativo, es decir, el mismo núcleo de funcionamiento del procesador y periféricos básicos, pero con variantes en ciertos módulos avanzados, encapsulados y la cantidad de terminales. Se consideraba muy importante la experiencia de conectar una memoria de programa y una memoria de datos, pero sin que la conexión se complicara como si se utilizara un microprocesador de 16 o 32 bits. El curso utilizando el 8051 se mantuvo hasta mayo de 2023, con los mismos antecedentes, pero cambiando significativamente los cursos subsecuentes y con la necesidad de reestructurar este curso básico de microprocesadores. Cantidad, ubicación y contenido de cursos sobre Microcontroladores en otros programas.

En la actualidad y particularmente en el ITESO, tres programas de Ingeniería, a nivel licenciatura, desarrollan competencias para el diseño y desarrollo de aplicaciones utilizando microcontroladores: Ingeniería Electrónica, Ingeniería en Mecatrónica e Ingeniería en Sistemas Digitales Embebidos.

En el caso de Ingeniería en Electrónica e Ingeniería en Sistemas Digitales Embebidos los cursos asociados a Microcontroladores son: Fundamentos de Microcontroladores y Microprocesadores, Sistemas Embebidos basados en microcontroladores, Sistemas operativos en tiempo real y Redes para sistemas embebidos.

Para Ingeniería en Mecatrónica los cursos son: Fundamentos de Microprocesadores y Microcontroladores e Integración de Sistemas Embebidos.

Es difícil y complejo, desde el punto de vista didáctico, cubrir la amplia variedad de microcontroladores y sus diferentes enfoques. Lo ideal, en experiencia del autor, sería estudiar microcontroladores pequeños y económicos, con suficientes recursos de hardware para aplicaciones muy simples en un primer curso. Microcontroladores de tamaño medio, 40 a 64 pines para otro rango de aplicaciones, usando librerías en un segundo curso. Microcontroladores multi núcleo, programados con o sin sistema operativo, microcontroladores con unidades especializadas para el Procesamiento Digital de Señales (PDS), en un tercer curso. Lo anterior solo en el concepto

de microcontroladores. Para otros casos, en específico Microprocesadores, se requerirían, un curso al menos, deseable dos.

En otras universidades donde se ofertan carreras como Ingeniería Biomédica, Ingeniería en Energías renovables, Ingeniería en Robótica, por mencionar algunas, también llevan al menos un curso asociado a microcontroladores.

El microcontrolador se convierte en el corazón del sistema que adquiere información del entorno, la procesa y toma acciones sobre dispositivos externos al microcontrolador, además de tener la capacidad de comunicarse con otros dispositivos cercanos o lejanos, en la misma tarjeta. El microcontrolador le da el término de inteligente al dispositivo, con algoritmos adecuados a la problemática y al entorno del sistema embebido.

2.6. Relación con otros cursos en los programas de ingeniería: antecedentes, simultáneos y sucesores.

Es común que antes del primer curso de microprocesadores o microcontroladores se coloque un curso de Fundamentos de Sistemas Digitales o también llamado simplemente Diseño Digital, donde se revisan los conceptos de electrónica digital combinacional y secuencial.

Para un curso inicial de microcontroladores es muy importante que el estudiante tenga la capacidad de entender la información que viene descrita en el *Reference Manual* elaborado por el fabricante del dispositivo. En concreto es común que el fabricante exponga diagramas donde muestra el comportamiento del microcontrolador. Dichos diagramas incluyen módulos digitales tales como: compuertas lógicas, multiplexores, *flip flops*, etc.

Por otro lado, también como antecedente, es muy deseable que el estudiante haya aprobado un curso de programación. Actualmente, es común en varias universidades que el lenguaje de programación inicial en las carreras de ingeniería sea Python. Una de las características de Python es que puede utilizarse en diversos campos, es comparativamente fácil de aprender y dispone de un conjunto grande de librerías tanto para aplicaciones básicas como para aplicaciones de mayor complejidad.

En el desarrollo profesional de sistemas embebidos se utiliza normalmente lenguaje C la variante de C++. Por lo que se considera altamente deseable que dicho lenguaje se estudie de manera previa al curso inicial de microcontroladores.

En el microcontrolador tenemos integrado en un solo circuito integrado los elementos constitutivos más importantes: CPU, memorias de programa y datos, periféricos integrados. A juicio del autor, en tiempos actuales se considera recomendable empezar con un microcontrolador básico y dejar para cursos posteriores el manejo de microprocesadores que contengan mayores recursos tales como memoria externa de programa y datos, periféricos de mayor complejidad o el uso de algún sistema operativo.

Uno de los cursos con los que se puede trabajar elementos comunes al estudio del primer microcontrolador es el de Instrumentación Electrónica. La relación se puede establecer en dos escenarios: En el primero bajo el supuesto que se tiene un sensor cuya salida es analógica, donde es necesaria una etapa de acondicionamiento de señal para amplificar, ajustar niveles de voltaje y filtrar la señal antes de ser convertida a señal digital para su procesamiento. El microcontrolador se encargará de controlar el muestreo de la señal y la conversión analógica a digital. El mismo microcontrolador puede continuar con tareas de filtrado, compensaciones a temperatura, etc.

En el caso de que el sensor sea digital, segundo escenario, el microcontrolador podrá recibir la información del sensor vía protocolos como IIC (*Inter Integrated Circuit*) o SPI (*Serial Peripheral Interface*).

El curso de Instrumentación Electrónica puede presentarse como antecedente, al mismo tiempo o después del primer curso de Microcontroladores. En cada caso hay ventajas y desventajas.

Otro de los cursos es el asociado a Control Automático Digital. Donde se busca, además de los conocimientos teóricos, la implementación de sistemas de control en ambientes embebidos utilizando un microcontrolador. En estos casos el microcontrolador recibirá las entradas al sistema, las procesará a partir del algoritmo de control y generará las señales necesarias para los actuadores del sistema. Dependiendo de la complejidad del algoritmo de control y de las consideraciones del sistema completo, el primer microcontrolador que se estudie puede ser suficiente para generar la solución embebida, entiéndase leer de sensores con salida analógica o digital, implementar el algoritmo de control y generar la salida necesaria para el control de la planta.

Otros cursos posteriores que pueden beneficiarse de los microcontroladores son: electrónica de potencia, comunicaciones digitales, robótica, desarrollo de proyectos en ambientes embebidos, etc.

2.7. Tipo de Microcontroladores utilizados.

En diversas universidades en México, incluyendo al ITESO, se han utilizado, para el primer curso de microcontroladores, diversos fabricantes y familias.

En los años 90s se utilizaban las familias HC05 y HC11 de Motorola. Microcontroladores de 8 bits, programados en lenguaje ensamblador, ya que los compiladores para el lenguaje C no eran gratuitos.

Merece un comentario aparte el fabricante Microchip con sus familias PIC. Los PIC12 se usaron a finales de los años 90's en diversas universidades. Herramientas gratuitas para editar, compilar, cargar el código y ambientes de depuración, hacían de manera gratuita el ambiente de desarrollo completo. Además de un conjunto de códigos de ejemplo y librerías bastante poderosas, todo en lenguaje ensamblador. Lo anterior fue de tal impacto, que se escribieron muchos libros para aprender a usar el microcontrolador y se siguen estudiando hasta la fecha.

En los años 2000s se popularizaron en algunas universidades familias de Motorola/Freescale HC08 y HCS08 todavía de 8 bits, pero con mejoras significativas. En estos años es cuando el ITESO gana en un periodo de tres años, dos concursos de NXP para Norteamérica. Compitiendo con más de 800 equipos, anuales, entre universidades y centros de investigación.

Mención especial tiene la influencia de la plataforma Arduino [7]. Esta plataforma es una combinación de una placa o tarjeta electrónica con un microcontrolador, software para el microcontrolador basado en librerías y software auxiliar para computadora personal. La plataforma permite poder usar el microcontrolador sin necesidad de un conocimiento detallado del mismo. Surge como una plataforma de bajo costo para desarrollar soluciones y después evoluciona para convertirse en un ambiente de desarrollo de prototipos rápidos para "hobistas" o *makers*. Dado que no es necesario conocer mucho de las características del microcontrolador, no se considera adecuada para soluciones profesionales. La plataforma Arduino no tiene un esquema de depuración de uso generalizado, elemento muy importante en soluciones profesionales. En [8] se presenta la evolución de un curso de microcontroladores bajo la influencia de Arduino, de dónde se toman elementos metodológicos que se consideran valiosos, como el desarrollo de múltiples ejercicios prácticos.

En una línea similar a Arduino se encuentra la plataforma Raspberry. Entre las diferencias más significativas están un microcontrolador más avanzado y, por lo tanto, un tipo de aplicaciones con mayor complejidad en cómputo, así como interfaces más demandantes en velocidad y procesamiento. Al igual que con Arduino, muchas universidades en el mundo, tanto para nivel de Licenciatura como de Maestría, usan estas plataformas. En algunos casos sin desarrollar experiencia fuerte en el uso de los microcontroladores sin librerías. Y es en estos casos cuando la industria se ve afectada por la falta de competencias en microcontroladores.

La cantidad de recursos de cómputo ha crecido en las aplicaciones y los costos de los microcontroladores se han reducido, de tal forma que hoy se utilizan microcontroladores de 32 bits. Mucha más memoria de programa y datos, mayor cantidad y funcionalidades de periféricos integrados. Pero del otro lado de la balanza se encuentra la situación de los encapsulados: PLCC (*Plastic Leaded Chip Carrier*) o TSSOP (*Thin Shrink Small Outline Package*), por mencionar algunos, donde el diseño del PCB's a la medida del problema se complica considerablemente.

Mención especial merece la documentación que ofrece el fabricante del microcontrolador. Una habilidad importante que debe de desarrollarse en los estudiantes es poder entender el comportamiento a partir de la información de los manuales [9]. Ya que el uso de librerías no permite usar el microcontrolador a toda su capacidad y de manera eficiente. Las librerías que desarrolla el fabricante tienen las funcionalidades más comunes, pero no toda la capacidad del componente. No forzosamente hay libros para aprender a usar cualquier microcontrolador, por lo que poder entender el comportamiento desde un manual es muy importante. El problema es que los manuales no los escribe el fabricante pensando en que el lector aprenderá su primer microcontrolador con la lectura. La estructura de cada capítulo del manual es bajo el supuesto que ya se leyó el resto del manual. Lo que no tiene lógica, pero es la realidad. Todo esto lleva a que un criterio importante para seleccionar el microcontrolador es la calidad del manual de referencia.

Normalmente el profesor del curso tiene que elegir la estrategia: extraer secciones del manual y desarrollar su propio material, sin obligar al estudiante a leer el manual y así tener un esquema más didáctico o, el otro extremo que es acompañar al estudiante en la lectura directa del manual con una guía detallada de que leer y que no en cada momento específico.

Considerando el uso de microcontroladores en la industria tratando de elegir cual fabricante es mejor y cual componente específico es el más usado, no hay una respuesta absoluta. La totalidad de los microcontroladores tienen segmentos de mercados y aplicaciones donde unos son mejores

que otros y viceversa. Mientras que en la actualidad la mayor parte de soluciones, en volumen, manejan microcontroladores de 32 bits, todavía hay casos para usar microcontroladores de 8 o 16 bits. Respecto a esta problemática en particular conviene revisar algunas aportaciones como en [10]. Donde a partir de varios criterios de selección, presentan un proceso para decidir sobre el microcontrolador a usar en una aplicación.

Tratando de situar la selección del microcontrolador al uso de estos en el entorno de aplicaciones desarrollados en México podemos hablar de algunos fabricantes, no de manera exhaustiva: NXP Renesas, Microchip, Infineon, STM, Texas Instruments, Analog Devices, Dallas Semiconductor, Atmel, Cypress, NEC.

El abanico de microcontroladores es muy grande en la industria, tanto en la cantidad de fabricantes como el nivel de complejidad. Esta diversidad fortalece la necesidad de que se aprenda la teoría, los principios de operación y se exponga al estudiante a los esquemas de desarrollo en la industria, de tal manera que pueda usar una variedad importante de microcontroladores, de diversos fabricantes en diversas aplicaciones.

En el tema específico de Inteligencia Artificial (IA), es común encontrar el término *tinyML* para el uso de IA en soluciones embebidas. Otro término similar es *Edge AI*, donde se considera microcontrolador Edge a aquél que está en la última etapa, directa al sensor y/o actuador, normalmente de bajo costo y baja capacidad de cómputo. La parte asociada al procesamiento masivo de datos se genera en otro microcontrolador o microprocesador de mayores capacidades o en el caso general, en la nube.

Ante el escenario descrito, el estudio en este ToG busca encontrar la relación adecuada entre los principios teóricos básicos del uso de microcontroladores, los principios de interconexión del microcontrolador con diversos tipos de entradas y salidas. La comunicación con los protocolos básicos: UART, IIC y SPI. Además de las estructuras de código y algoritmos para el desarrollo adecuado de software para soluciones embebidas.

3. Propuesta del Curso

3.1. Consideraciones iniciales

El curso se ofrece para estudiantes que hayan aprobado el curso de Fundamentos de Sistemas Digitales, donde se aprenden las bases del álgebra booleana y se desarrollan proyectos de baja-mediana complejidad relacionados con electrónica digital utilizando un lenguaje de descripción de hardware (Verilog) y tarjetas de desarrollo con dispositivos programables (FPGA's).

Siendo el primer curso donde se aprende a utilizar un microcontrolador de manera profunda, se prioriza las competencias en el nivel de entender y usar adecuadamente los fundamentos de los microcontroladores, tanto en la programación como en la interconexión con otros dispositivos necesarios en un sistema embebido. En [11] se presenta una estrategia similar a la sugerida para el curso de Fundamentos de Microprocesadores y Microcontroladores.

El microcontrolador seleccionado es un MSP430G2553 de la empresa TI [12]. Los elementos principales para seleccionarlo fueron:

I. Suficientemente simple (poca cantidad de periféricos y de poca complejidad, pocas terminales) para ser el primer microcontrolador.
II. Con suficiente capacidad de cómputo (CPU de al menos 16 bits y con un conjunto de instrucciones típicas como multiplicación en hardware).
III. Periféricos integrados: GPIO, *timers*, interrupciones, ADC, UART, SPI, IIC, entradas y salidas temporizadas.
IV. Bajo costo tanto del microcontrolador como de la tarjeta de desarrollo.
V. Encapsulado DIP para poder generar PCB de baja complejidad de fabricación.
VI. Tarjeta electrónica que incluya al microcontrolador y los elementos necesarios para poder descargar el código y poder hacer actividades de depuración.

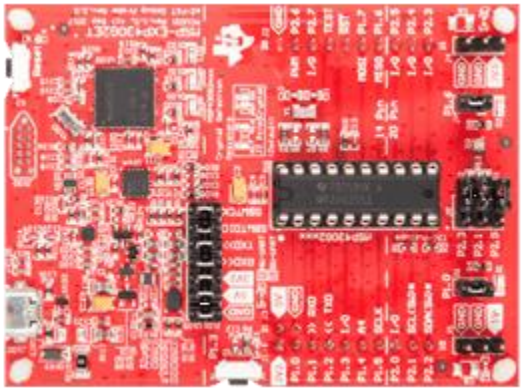

Figura 5 Tarjeta electrónica MSP430G2ET LaunchPad [12]

La Figura 5 muestra la tarjeta electrónica que incluye el microcontrolador MSP430G2553

3.2. Presentación de la asignatura.

El curso integra competencias desarrolladas en cursos anteriores como Fundamentos de Sistemas Digitales y Programación Estructurada.

Se desarrollan competencias en la programación e interconexión de Microcontroladores con sensores, actuadores y módulos de comunicación. La programación se hace directamente sobre los registros de configuración del microcontrolador, evitando, en general, el uso de librerías (SDK o BSP). Por lo que se define como necesaria la lectura constante del Manual de Referencia del Microcontrolador. En cursos posteriores se hará uso de librerías y sistema operativo.

El lenguaje de programación es ANSI C enfocado a ambientes embebidos. Ya que es lenguaje más usado en aplicaciones embebidas con microcontroladores.

3.3. Propósito general.

El alumno será capaz de diseñar e implementar las interfaces y programar un microcontrolador para cumplir los requerimientos de un sistema que incluya sensores analógicos, sensores digitales, actuadores y módulos de comunicación. Además de poder implementar en el

microcontrolador algoritmos básicos para poder cumplir requerimientos de funcionamiento y desempeño en sistemas embebidos.

3.4. Propósitos específicos.

En este curso se busca que el alumno sea capaz de:

(1) Comprender las características de un microcontrolador y pueda usarlo en diferentes aplicaciones.
(2) Utilizar el módulo ADC (Analog to Digital Converter) del microcontrolador para interconectar sensores con salida de señal analógica.
(3) Controlar motores de DC (Direct Current) y motores a pasos cumpliendo con requerimientos específicos.
(4) Interconectar dispositivos y desarrollar programas usando el protocolo serial asíncrono para comunicarse con computadoras, teléfono celular, o cualquier otro dispositivo que use dicho protocolo.
(5) Interconectar dispositivos y desarrollar programas usando el protocolo IIC (Inter Integrated Circuit) para comunicarse con sensores, actuadores, pantallas LCD (liquid cristal display) y cualquier otro dispositivo que use dicho protocolo.
(6) Interconectar dispositivos y desarrolla programas usando el protocolo SPI (Serial Peripheral Interface) para comunicarse con sensores y actuadores o cualquier otro dispositivo que use dicho protocolo.
(7) Generar interfaces en hardware y la programación necesaria para manejar entradas y salidas temporizadas.
(8) Utilizar de manera adecuada el analizador lógico para verificar el funcionamiento de ciertas interfaces.
(9) Utilizar adecuadamente las herramientas de control de versiones de código.

3.5. Contexto curricular.

El trabajo en esta asignatura requiere de competencias teóricas y habilidades para la programación, así como para la interconexión de dispositivos electrónicos.

Para la correcta lectura de los Manuales de Referencia, son necesarios los conocimientos desarrollados en el curso de Fundamentos de Sistemas Digitales.

La experimentación que se desarrolla emplea como plataforma un microcontrolador y diversos dispositivos electrónicos y eléctricos.

Se realiza trabajo experimental usando el ambiente IDE (Integrated Development Environment) del microcontrolador seleccionado por el profesor en acuerdo con la academia. Este software puede ser instalado en la computadora personal del estudiante de manera gratuita.

En [13] se presenta un conjunto de experimentos sugeridos para ir desarrollando, poco a poco, el trabajo práctico en el aprendizaje del microcontrolador. Los ejercicios del curso sugerido de Fundamentos de Microprocesadores y Microcontroladores toman una estrategia similar. Esta asignatura es base para lograr los objetivos de aprendizaje de Sistemas Embebidos basados en Microcontroladores, además de aportar elementos para el curso de Sistemas de Control Automático.

En la Figura 6 se muestra un diagrama con los cursos de Ingeniería Electrónica que se relacionan de manera estrecha con el curso de Fundamentos de Microcontroladores y Microprocesadores.

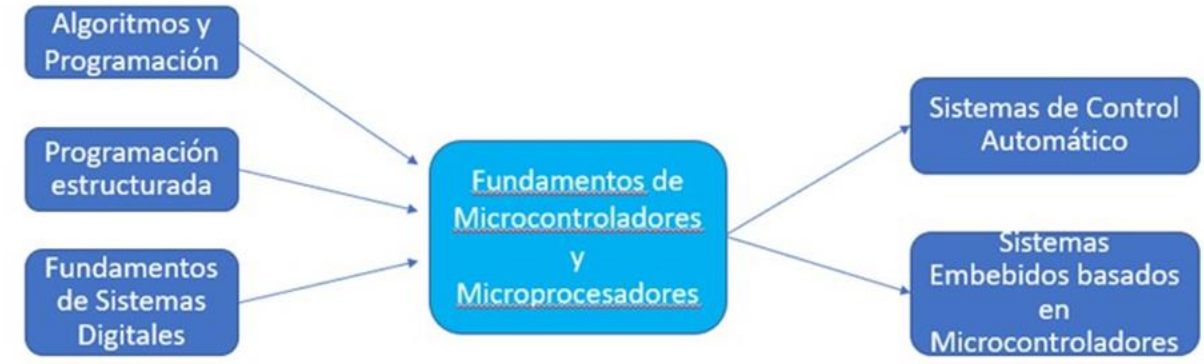

Figura 6 Relación del curso con otras asignaturas

3.6. Contenido del curso.

1 Conceptos Básicos y Características de los Microcontroladores (Sesiones:1)

1.1 Definición de Microprocesador, Microcontrolador y Sistema Embebido.

1.2 Clasificación inicial de los Microcontroladores.

1.3 Lenguajes de Programación para Microcontroladores.

1.4 Interconexiones básicas: sensores, actuadores, comunicaciones.

1.5 Características de Sistemas embebidos basados en Microcontroladores.

1.6 Arquitectura del microcontrolador seleccionado.

2 Pines entrada y salida de propósito general (GPIO) (Sesiones 3).

2.1 Multifuncionalidad de los pines.

2.2 Diagrama esquemático de la tarjeta de desarrollo.

2.3 Configuración de pines como GPIO y selección de entrada o salida.

2.4 Primer programa en lenguaje C para el microcontrolador (*blinking led*).

2.5 Tool chain: editor, compilador, loader, debugger.

2.6 Uso de analizador de estados lógicos para visualizar la secuencia de manejo para el motor a pasos.

3 *Timer* básico (Sesiones 2).

3.1 Concepto de *timer*: base de tiempo, rango de conteo, preescaler, banderas.

3.2 Generación de tiempos variables usando *timers*. Cálculo de tiempos a partir de los ticks del *timer*.

3.3 Múltiples *timers*.

3.4 Caso de uso: control de motor a pasos bidireccional.

4 Interrupciones (Sesiones 2).

4.1 Conceptos de interrupciones: banderas, habilitadores locales, deshabilitadores globales, vector de interrupción, ISR (Interrupt Service Routine), prioridades, anidamiento de interrupciones, respaldo y restauración de contextos, retorno de interrupción.

4.2 Registros específicos del microcontrolador para configurar y manejar interrupciones.

4.3 Estructuras de código para el manejo de interrupciones.

4.4 Caso de uso de *Timer* básico con interrupciones. Para generar tiempos en la secuencia de pasos de un motor *stepper*.

5 Manejo de señales analógicas ADC (Analog to Digital Converter) (Sesiones 3).

5.1 Conexión de un sensor a un microcontrolador.

5.2 Entradas y salidas en un ADC.

5.3 Resolución en un ADC.

5.4 Principio de operación de un ADC de aproximaciones sucesivas.

5.5 Resumen de conceptos importantes en un ADC: Tiempo de conversión, resolución, tipo de convertidor, *Sample and hold*, referencia común o diferencial, Voltaje de referencia.

5.6 Pines que pueden funcionar como entrada de ADC. Multiplexor analógico.

5.7 Diagrama del ADC interno del microcontrolador.

5.8 Modos de operación del ADC. Lectura de un potenciómetro y lectura de dos potenciómetros.

6 Comunicación tipo UART (Sesiones 4).

6.1 Comunicaciones: paralela vs serial, serial síncrona vs serial asíncrona.

6.2 Diagrama interno de una UART.

6.3 Configuración del reloj para definir *bit rate* (*baud rate*). Tolerancias.

6.4 Transmisión de datos. Convertidor itoa (integer to ascii). Recepción de datos en Matlab, Phyton y teléfono celular (*Bluetooh*).

6.5 Recepción de datos. Convertidor atoi (ascii to integer). Transmisión de datos en Matlab, Phyton y teléfono celular (Bluetooth).

6.6 Uso del analizador de estados lógicos para visualizar una trama de comunicación tipo UART.

7 Comunicación tipo IIC (Sesiones 4).

7.1Estructura de la trama de comunicación.

7.2 Implementación con pines de GPIO (IIC virtual).

7.3 Configuración de reloj.

7.4 Comunicación con sensor o actuador digital (IMU).

7.5 Uso del analizador de estados lógicos para visualizar una trama de comunicación tipo IIC.

8 Comunicación tipo SPI (Sesiones 2).

8.1 Diagrama interno básico del módulo SPI.

8.2 Configuración de reloj.

8.3 Comunicación con sensor o actuador digital.

8.4 Uso del analizador de estados lógicos para visualizar una trama de comunicación tipo SPI.

9 Entradas y salidas temporizadas (Sesiones 4).

9.1 Input Capture (IC).

Medición de tiempos: anchos de pulso y frecuencia.

9.2 Output Compare (OC).

Generación de modulación por ancho de pulso, *Pulse Width Modulation* (PWM)

Generación de ondas cuadradas ad-hoc.

Control de velocidad de motor DC con PWM.

9.3 Pulse Width Modulation (PWM).

Edge Aligned/Center Aligned.

Death band.

Control de velocidad de motor AC con PWM.

9.4 Uso del analizador de estados lógicos para visualizar una señal tipo PWM.

3.7. Referencias bibliográficas para el curso

A partir del análisis bibliográfico realizado se ha determinado utilizar las referencias bibliográficas mostradas en la Tabla 1 que apoyarán al estudiante en el entendimiento, programación y transferencia de conocimiento del microcontrolador de TI seleccionado:

1	**Embedded Systems Design Using the TI MSP430 Series** Nagy, Chris 2003, Elsevier Science & Technology (Newnes) ISBN 9780750676236
2	**MPS430 Microcontroller Basics** Davies, John H. 2012, Elsevier Science & Technology ISBN 9789380501857
3	**Microcontroller Programming and Interfacing. Texas Instruments MSP430** Barrett, Steven & Pack, Daniel J. 2011 MORGAN & CLAYPOOL PUBLISHERS ISBN 9781608457137

Tabla 1 Bibliografía del curso

3.8. Criterios de evaluación.

El logro de los propósitos específicos del curso será evaluado a través de los siguientes elementos y su ponderación correspondiente:

Ejercicios de puesta en marcha de los periféricos	20 %
Proyecto de integración	20 %
Trabajos experimentales (prácticas)	20 %
Exámenes parciales	40 %

Para aprobar el curso se necesita tener un promedio mínimo de 60 en los exámenes.

Siendo el objetivo del curso la promoción del desarrollo de ciertas competencias, a lo largo del mismo, es necesario que todas las actividades se realicen en los momentos indicados y se entreguen antes de la hora y fecha establecidas, tiempo administrado por la plataforma del curso, por lo que no se aceptarán entregas tardías ni habrá posibilidad de recuperación. Igualmente, en el caso del producto de alguna actividad que se identifique que tiene elementos copiados del producto de otro compañero, ambos productos serán anulados.

3.9. Estrategia didáctica

Se intensionará el desarrollo de las competencias con las siguientes experiencias de aprendizaje: ejercicios, prácticas y proyecto integrador.

Los **ejercicios** de puesta en marcha de los periféricos consisten en hacer interconexiones básicas y modificar códigos generados en clase para experimentar con los módulos del microcontrolador. En [14], se presenta un artículo donde se presenta un entrenamiento, del tipo *Hands-on* en sistemas embebidos. Referente para las actividades específicas llamadas ejercicios.

El **proyecto integrador** busca desarrollar el hardware y software necesario para resolver un problema cercano a la realidad. El estudiante puede revisar un catálogo de proyectos propuesto por la Universidad de Cornell, que se mantiene en un espacio WEB abierto, donde se presentan los proyectos realizados en su primer curso de microcontroladores [15]. En [16] se presenta el diseño de una práctica progresiva, como se propone en este curso, al igual que el desarrollo de una experiencia integrada y realista asociada a la profesión.

En [17] se presenta la implementación del diseño de un sistema embebido como proyecto final estudiantil usando aprendizaje basado en problemas. De ahí se toman elementos para el desarrollo de la propuesta como establecer un proyecto que se va desarrollando con diferentes ejercicios prácticos durante todo el curso. Además de experimentar con otros elementos del microcontrolador que no son requeridos para el proyecto específico.

En [18] se presentan varios elementos a considerar para el desarrollo de los escenarios prácticos de un curso orientado a ejercicios de laboratorio. Específicamente se plantean algunas competencias no técnicas, como el trabajo en equipo y la innovación.

Cada **práctica** integra un conjunto de módulos periféricos con un programa que los pone a funcionar juntos para una aplicación básica o un subsistema.

Los **exámenes** se realizarán durante el tiempo de una sesión de clase (1h40) y se entregarán a través de la plataforma del curso, CANVAS, por lo que deberá de respetarse estrictamente su duración, pues el sistema cerrará la posibilidad de entrega fuera del tiempo determinado.

3.10. Calendario del curso

Las Figuras 7 y 8 muestran la distribución de sesiones en un semestre de 16 semanas con frecuencia de dos sesiones por semana.

		Agosto					Septiembre								Octubre									Noviembre								Dic
		M	V	M	V	M	V	M	V	M	V	M	V	M	V	M	V	M	V	M	V	M	V	M	V	M	V	M	V	M	V	M	V
		15	18	22	25	29	1	5	8	12	15	19	22	26	29	3	6	10	13	17	20	24	27	31	3	7	10	14	17	21	24	28	1
	Temas / Sesiones =>	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**
1 Conceptos Básicos y Características de los Microcontroladores																																	
	1.1 Definición de Microprocesador, Microcontrolador y Sistema Embebido.																																
	1.2 Clasificación inicial de los Microcontroladores																																
	1.3 Lenguajes de Programación para Microcontoladores																																
	1.4 Interconexiones básicas: sensores, actuadores, comunicaciones																																
	1.5 Características de Sistemas embebidos basados en Microcontroladores																																
	1.6 Arquitectura del microcontrolador seleccionado																																
2 Pines entrada y salida de propósito general (GPIO)																																	
	2.1 Multifuncionalidad de los pines																																
	2.2 Diagrama esquemático de la tarjeta de desarrollo.																																
	2.3 Configuración de pines como GPIO y selección de entrada o salida																																
	2.4 Primer programa en lenguaje C para el microcontrolador (blinking led)																																
	2.5 Tool chain: editor, compilador, loader, debugger.																																
	2.6 Lenguaje C: variables																																
3 Timer básico																																	
	3.1 Concepto de timer: base de tiempo, rango de conteo, preescaler, banderas																																
	3.2 Generación de tiempos variables usando timers. Cálculo de tiempos a partir de los ticks del timer																																
	3.3 Múltiples timers																																
	3.4 Caso de uso: control de motor a pasos bidireccional																																
	3.5 Lenguaje C: arreglos de constantes, estructuras condicionales y de repetición. Funciones y paso de parámetros.																																
4 Interrupciones																																	
	4.1 Conceptos de interrupciones:banderas, habilitadores locales, deshabilitadores globales, vector de interrupción, ISR (Interrupt Service Routine), prioridades, anidamiento de interrupciones, respaldo y restauración de contextos, retorno de interrupción																																
	4.2 Registros específicos del microcontrolador para configurar y manejar interrupciones																																
	4.3 Estructuras de código para el manejo de interrupciones.																																
	4.4 Caso de uso de Timer básico con interrupciones																																
	4.5 Lenguaje C: Variables locales, variables globales, temporalidad y visibilidad de variables																																
	Examen																																

Figura 7 Calendario del curso

5 Manejo de señales analógicas	
	5.1 Convertidor Analógico a Digital (ADC). Tiempo de conversión, resolución, tipo de convertidor, Sample and hold, referencia común o diferencial, Voltaje de referencia
	5.2 Convertidor Digital a Análogo. Tiempo de establecimiento, voltaje de referencia, resolución, tipo de convertidor.
	5.3 Comparador Analógico. Configuraciones, histéresis, filtros
	5.4 Ejercicios: ADC, DAC, ACMP
	5.5 Lenguaje C: Arreglos de datos variables.
6 Comunicación tipo UART	
	6.1 Comunicaciones: paralela vs serial, serial síncrona vs serial asíncrona
	6.2 Diagrama interno de una UART
	6.3 Configuración del reloj para definir bit rate (baud rate). Tolerancias
	6.4 Transmisión de datos.Convertidor itoa(). Recepción de datos en Matlab, Phyton y teléfono celular (Bluetooh)
	6.5 Recepción de datos. Convertidor atoi(). Transmisión de datos en Matlab, Phyton y teléfono celular (Bluetooh)
	6.6 Lenguaje C: Punteros, punteros a arreglos.
7 Comunicación tipo IIC	
	7.1Estructura de la trama de comunicación
	7.2 Implementación con pines de GPIO (IIC virtual)
	7.3 Configuración de reloj
	7.4 Comunicación con sensor o actuador digital (IMU)
	7.5 Lenguaje C: Archivos tipo h (header files). Estructura de archivos. Principios de MCAL
8 Comunicación tipo SPI	
	8.1 Diagrama interno básico del módulo SPI
	8.2 Configuración de reloj
	8.3 Comunicación con sensor o actuador digital.
	8.4 Lenguaje C: Consideraciones de consumo de memoria (código y datos) y de tiempos de ejecución.
	Examen
9 Entradas y salidas temporizadas	
	9.1 Input Capture (IC). Medición de tiempos: anchos de pulso y frecuencia.
	9.2 Output Compare (OC). Generación de PWM, generación de ondas cuadradas ad-hoc. Motor DC
	9.3 Pulse Width Modulation (PWM). Edge Aligned/Center Aligned. Death band. Motor AC
	9.4 Lenguaje C: Compilador y linker

Figura 8 Calendario del curso (Continuación)

4. Unidades de aprendizaje

Una vez descritos los elementos del curso en el Capítulo 3, en este capítulo se presentan, con mayor detalle, las consideraciones técnicas y didácticas para desarrollar cada uno de los temas en las sesiones de clase, con el objetivo de que sean un apoyo para el profesor.

4.1. Conceptos básicos y características de los Microcontroladores.

A. Definición de Microprocesador, Microcontrolador y Sistema Embebido.

El estudiante asimila la diferencia entre el microprocesador y microcontrolador, tomando como elemento principal la memoria interna de programa, memoria interna de datos y los equipos periféricos internos.

Por otro lado, entiende la definición de un sistema embebido y ubica la gran cantidad y variedad de aplicaciones de sistemas embebidos. Distinguiendo las diferencias de aplicaciones asociadas al cómputo convencional: computadores personales y dispositivos móviles como *smartphones* y tabletas electrónicas.

B. Clasificación inicial de los Microcontroladores.

El estudiante reconoce los elementos que ayudan a clasificar los microcontroladores: tamaño de los datos en la ALU (Arithmetic Logic Unit), memoria interna de programa y datos, cantidad-complejidad-orientación de los dispositivos de entrada y salida.

C. Lenguajes de programación para microcontroladores.

El estudiante conoce las diferencias entre los diferentes lenguajes de programación y su nivel de abstracción: lenguaje máquina, lenguaje ensamblador, lenguaje C y otros lenguajes como Micropython. Encontrando en cada uno sus

características de complejidad, tiempo de desarrollo, eficiencia del uso de la memoria de código y de la memoria de datos.

D. Interconexiones básicas: sensores, actuadores, comunicaciones.

El estudiante conoce diferentes aplicaciones de sistemas embebidos y reconoce los sensores, actuadores y comunicaciones presentes en dichos sistemas. Es importante mencionar un abanico amplio de aplicaciones: médicas, automotrices, productos en el hogar, dispositivos especializados para muchas profesiones, por mencionar solo algunas.

E. Características de Sistemas embebidos basados en Microcontroladores.

En una tabla comparativa, enlista diferencias tecnologías usadas en sistemas electrónicos: microcontroladores, PLCs y computadoras especializadas. Los parámetros por comparar son: costo del sistema, consumo de energía, espacio físico, factor de forma, costo de diseño de software, costo y tiempo de diseño de hardware, capacidad de cómputo.

F. Arquitectura del microcontrolador seleccionado.

El estudiante entiende, a grandes rasgos, los elementos constitutivos del microcontrolador seleccionado para el curso y localiza dicha información en el manual de referencia. En específico la capacidad de la Unidad Central de Proceso (CPU), memoria interna de programa, memoria interna de datos y periféricos integrados.

4.2. Terminales de entrada y salida de propósito general.

A. Multifuncionalidad de los pines (terminales).

Inicialmente se presenta la distribución de los pines y la manera en que se enumeran. También se presenta cómo cada terminal tiene varias funcionalidades y

la manera en que conforme se vaya avanzando en el curso se irán entiendo una por una.

Se conoce la posición de las terminales principales de Alimentación (*Power Supply*) y tierra (GND). Se presenta el rango de alimentación del microcontrolador y como puede proporcionarse dicho voltaje vía la conexión USB con la computadora, adecuando el voltaje vía un regulador presente en la tarjeta.

Se comenta la situación común, en muchos microcontroladores, que pueden trabajar con un oscilador interno y con un circuito Power On Reset (POR) interno.

También se presentan las conexiones que ofrece la tarjeta de desarrollo entre el microcontrolador y las tiras de pines, que permiten fácilmente hacer conexiones hacia el microcontrolador. Se refuerza el tema de la multifuncionalidad de los pines.

B. Diagrama esquemático de la tarjeta de desarrollo.

Se presentan los diagramas esquemáticos de la tarjeta de desarrollo, directamente del manual de la tarjeta y de un archivo que incluye exclusivamente el diagrama.

Es importante que los estudiantes consulten las fuentes originales de la información. Extraer la información selecta a otras plataformas no se considera adecuado en esta etapa de aprendizaje.

También se presentan algunos elementos de la tarjeta de desarrollo además del microcontrolador. Los LEDs y *push button* disponibles para que el usuario pueda probar códigos. Por otro lado se presenta la sección de tarjeta que contiene la circuitería auxiliar para poder descargar el código desde la computadora (*loader*), para hacer tareas de depuración (*debugger*) y tener comunicación serial en un profile de USB.

Se deben de explicar las funcionalidades de los siete *jumpers* de operación que comparte la sección complementaria de la tarjeta con la sección del microcontrolador, aunque sea una descripción básica.

C. Configuración de pines como GPIO y selección de entrada o salida.

Se considera importante un repaso respecto a parámetros eléctricos de los pines de un dispositivo digital: I_{OH}, I_{OL}, I_{IH}, I_{IL}, V_{OH}, V_{OL}, V_{IH}, V_{IL}. De tal manera que sean tomados en cuenta al momento de conectar elementos básicos como LEDs o *switches*.

Se revisan algunos parámetros eléctricos del microcontrolador tomados directamente del *Data Sheet*.

Se realiza un análisis de la configuración por default de los pines usados como propósito general GPIO (General Purpose Input Output) y la razón de por qué los pines bidireccionales se configuran como entrada después de un *reset* .

Se revisa el diagrama esquemático de los puertos del microcontrolador. Es importante que el estudiante tenga la habilidad de entender la manera en que se interconectan compuertas lógicas, multiplexores, *flip-flops*, etc; de tal manera que en conjunto den la funcionalidad requerida en los pines.

D. Primer programa en lenguaje C.

A partir de una guía en forma de pseudocódigo, se introduce a los estudiantes al primer programa. El código configura los elementos mínimos necesarios para poder prender y apagar un LED que está presente en la tarjeta de desarrollo.

E. Tool chain.

Se presenta todo el *tool chain* asociado al IDE (Integrated Development Environment): editor, compilador, *loader* y *debugger*.

Se realiza el proceso completo con el primer programa: se edita, se compila, se corrigen errores, se carga el programa a la tarjeta y se ejecuta usando diversas funcionalidades. De igual manera se ve el contenido de variables y se usan *breakpoints*.

4.3. *Timer* básico.

A. Concepto de *timer*: base de tiempo, rango de conteo, preescaler, banderas.

Es importante presentar el concepto básico de *timer* usando elementos de la vida común. Puede usarse un reloj despertador mecánico y después migrar el concepto a las alarmas de un teléfono celular. Estableciendo como en ambos casos se tiene un *timer* que se compara constantemente contra una o más referencias de tiempo. Lo que se define como el principo de funcionamiento de una salida temporizada.

Se recomienda no profundizar en las fuentes de reloj (*clock source*) asociadas al módulo del *timer*. Dejar los *clock source* por *default* y presentarlos como fijos al momento. Más adelante se estudiará con el detalle que se requiere.

El *timer* se configura en modo ascendente hasta el máximo valor disponible en el rango (modo continuo).

Se presenta el concepto de bandera (*flag*), como un indicador que algo importante sucede en los módulos periféricos al procesador. En el caso particular, la bandera se prende cuando el *timer* alcanza el valor deseado para cumplir con el tiempo requerido.

B. Generación de tiempos variables usando *timers*. Cálculo de tiempos a partir de los ticks del *timer*.

Es importante realizar ejericios para calcular el número de cuentas requerido para alcanzar los tiempos solicitados a partir de la frecuencia del reloj, la configuración del *preescaler* y la frecuencia del reloj. Entre los tiempos a calcular se proponen: tiempos cortos con alta resolución de conteo, tiempos largos con baja resolución de conteo, tiempos varios generados con diferentes opciones en el *preescaler*.

C. Múltiples *timers*.

También se considera importante hacer ejercicios poniendo a funcionar más de un *timer* a la vez; o un mismo *timer* generando más de un lapso de tiempo

requerido. En CCR0 del *timer0* se genera un tiempo. En CCR1 y CCR2 del *timer1* se generan un par de tiempos distintos al de CCR0.

D. Caso de uso: control de motor a pasos bidireccional.

Como caso de aplicación, se utiliza la generación de la secuencia para mover un motor a pasos. Generando los lapsos de tiempos entre pasos y por lo tanto la velocidad de giro del motor usando un *timer*. Este ejercicio se considera con una combinación interesante entre simplicidad, el uso en una aplicación concreta, con una alta probabilidad de un considerable reuso en diversos proyectos en semestres futuros.

E. Uso del analizador de estados lógicos

El correcto uso del analizador de estados lógicos es muy importante en un curso de Fundamentos de Microprocesadores y Microcontroladores. Es la manera de visualizar las señales físicas presentes en la interconexión del microcontrolador con otros dispositivos. Por lo que en diferentes sesiones del curso se utiliza para verificar el comportamiento de las señales digitales de entrada y/o salida del microcontrolador. Revisar los tiempos de duración y las secuencias necesarias.

4.4. Interrupciones.

A. Conceptos de interrupciones.

Se sugiere presentar, usando elementos de la vida cotidiana, los siguientes conceptos asociados a interrupciones: banderas, habilitadores locales, deshabilitadores globales, vector de interrupción, ISR (Interrupt Service Routine), prioridades, anidamiento de interrupciones, respaldo y restauración de contextos, retorno de interrupción.

Por ejemplo, en una llamada telefónica, tenemos la opción de deshabilitar la recepción de llamadas. Si estoy en una llamada y llega otra, puedo priorizar a quien

atender, etc. Términos que se manejan en el vocabulario de interrupciones en microcontroladores.

Se explica un diagrama lógico donde están presentes banderas, habilitadores locales, el controlador de interrupciones con sus habilitadores y analizadores de prioridades, el vector ID y la ISR (Interrupt Service Routine).

Es importante analizar, como experiencia inicial, la tabla donde se encuentran todas las interrupciones, con las banderas que las generan, las direcciones de memoria reservada y las prioridades.

B. Registros específicos del microcontrolador para configurar y manejar interrupciones.

Se estudia como caso la generación de interrupciones para el *timer* básico. Considerando todos los elementos, incluyendo el habilitador global de interrupciones, el habilitador local y el código necesario para la aplicación que debe colocarse en la ISR (*Interrupt Service Routine*).

C. Estructuras de código para el manejo de interrupciones.

El estándar del lenguaje C no define palabras reservadas ni estructuras de codificación para el manejo de interrupciones. Por lo que cada compilador define como estructurar las ISR (Interrupt Service Routine). En varios casos se utiliza la palabra reservada *pragma* para definir el inicio del esqueleto de una ISR y establecer la referencia con la dirección asignada en la tabla de vectores de interrupción.

D. Caso de uso del *Timer* básico con interrupciones.

Desarrollo de un código de referencia donde se inicialice un *timer*, se inicialice la interrupción del *timer* y en la ISR se haga un procesamiento básico; por ejemplo, encender y apagar un LED con un tiempo de espera configurado con la interrupción del *timer.* En la ISR se hace el cambio de estado del LED.

4.5. Manejo de señales analógicas.

A. Conexión de un sensor a un microcontrolador.

Explicar el diagrama y el flujo que incluye: sensor, acondicionamiento de señal, ADC, CPU. Tratando de usar los conocimientos previos en filtros y amplificadores operacionales. Lo anterior para el caso de que el sensor entregue una señal analógica.

B. Entradas y salidas en un ADC.

Se considera de mucha importancia que esté clara la relación de entradas y salidas en un ADC. La señal de entrada es una señal continua en el tiempo y con valores de voltajes continuos en el rango de operación del ADC, definido a partir de las referencias de voltaje. Las señales de salida es una palabra digital, en donde la longitud de la palabra está definida por el tamaño del convertidor ADC.

C. Resolución en un ADC.

Usando ADCs de 8, 10, 12 y 16 bits. Desarrollar el concepto de resolución en un ADC. De igual manera utilizar los voltajes de referencia para definir el tamaño de paso.

D. Principio de operación de un ADC de aproximaciones sucesivas.

Se recomienda usar diferentes estrategias que ejemplifiquen el uso de búsquedas binarias en diferentes aplicaciones como el algoritmo de bisección para encontrar la raíz de una curva o encontrar un elemento en una lista ordenada.

Es importante explicar con detalle el principio de operación de un ADC de aproximaciones sucesivas y su respectivo árbol de generación de valores, relacionándolo con el algoritmo de búsqueda binaria.

E. Resumen de conceptos importantes en un ADC.

Una vez entendidos los puntos anteriores, se recomienda este momento donde en resumen se revisan los conceptos básicos de un ADC.

F. Pines que pueden funcionar como entrada de ADC. Multiplexor analógico.

Utilizando como referencia el *Data Sheet*, encontrar claramente los pines que pueden funcionar como entrada analógica y la manera de configurarlos.

G. Diagrama del ADC interno del microcontrolador

Utilizando el diagrama a bloques del ADC interno. Encontrar los elementos más importantes y entender la forma en que opera el ADC. No es indispensable que en la primera aproximación al tema se analice todo el diagrama. En el momento en que se revisa la documentación de los registros de configuración se puede ir profundizando poco a poco en diversos casos de uso.

H. Modos de operación del ADC.

Es común que los ADC internos en los microcontroladores tengan diferentes modos de operación. Es importante el principo de operación de dichos modos y la forma de configurarlos.

Es importante desarrollar códigos en diferentes modos de operación para poder terminar de entender los registros de configuración, de operación y aplicaciones para ellos.

4.6. Comunicación tipo UART (*Universal Asyncronous Receiver and Transmitter*).

A. Comunicaciones: paralela vs serial, serial síncrona vs serial asíncrona.

Muy probablemente es la primera ocasión en que los estudiantes entran al tema de las comunicaciones desde la perspectiva del diseño de sistemas, por lo que es importante iniciar con algunas definiciones.

Se presenta la foma de un diagrama de tiempo (trama) de comunicación tipo UART: *idle state*, *start* bit, bits de datos, opcionalmente el bit de paridad y finalmente el o los *stop* bits.

El diagama donde se interconecta el core (procesador) con la UART, los principales módulos internos en la UART, el reloj del módulo y las señales TxD y RxD. Además de varias banderas que indican el estatus y comportamiento de la UART.

Es importante entender los registros presentes en la UART y la forma en que se construyen con *flip flops* y circuitería combinacional. Los registros son de los siguientes tipos: PIPO (Parallel Input Parallel Output), SIPO (Serial Input Parallel Output) y PISO (Parallel Input Serial Output).

B. Diagrama interno de una UART

A partir del diagrama interno presentado en el *Reference Manual*, se analiza y detectan los elementos principales de la UART y la forma en que se interconectan. Se recomienda poner énfasis en los *shift registers* y el uso del reloj que define la velocidad de comunicación.

C. Configuración del reloj para definir bit *rate* (*baud rate*). Tolerancias

El fabricante del microcontrolador, Texas Instruments en este caso, define un periodo de muestreo de la señal de entrada a una velocidad 16 veces mayor a la definida en el *bit rate*.

El baud rate se configura seleccionando el reloj de la UART y diviéndolo entre el baud rate deseado. El resultado de la divisón se carga en los registros de baud rate.

Se revisa la forma en que debe de especificarse el reloj de la UART en el microcontrolador para lograr el bit *rate* deseado. Es importante hacer el análisis del máximo error permitido en la configuración del reloj, ya que en muchos casos el valor calculado no será un entero.

D. Transmisión de datos. Convertidor itoa(). Recepción de datos en Matlab, Phyton y teléfono celular (Bluetooh).

Es importante el uso del analizador lógico para mostrar la trama de comunicación.

Se presentan dos opciones de capa física para la comunicación. En una se usa el profile de USB para comunicación tipo UART, de tal manera que el dato sale/entra al microcontrolador de manera serial y se comunica a un microcontrolador auxiliar en la tarjeta de desarrollo. Este microcontrolador genera los *frames* USB en un profile de UART. Los datos físicamente llegarán a la computadora vía USB pero el protocolo detecta que vienen descritos como datos de UART y los presentan así al sistema operativo, para ese momento, la computadora los interpreta, en cualquier aplicación, como si hubieran llegado físicamente vía UART.

En la segunda opción de capa física se usa un módulo HC06 Bluetooth [19] a serial. De esta forma se puede recibir la información de forma inalámbrica en una computadora, un teléfono celular o una tableta electrónica. Es valioso dedicar el tiempo a revisar los comandos AT básicos en el módulo HC06.

Es importante algún ejercicio demostrativo donde se manden o reciban datos desde una computadora usando Matlab o Python y en un celular haciendo una aplicación usando el ambiente App Inventor, por ejemplo.

4.7. Comunicación IIC (*Inter-Integrated Circuit*).

A. Estructura de la trama de comunicación.

Iniciando con el planteamiento inicial del protocolo. Un maestro, múltiples esclavos direccionados por la propia trama de comunicación. Comunicación serial síncrona donde el maestro genera y comparte el reloj de los datos.

En el segundo momento se revisa la trama (*frame*) completa definida en el protocolo. Considerando cuatro escenarios: escritura de un dato, escritura de una secuencia de datos, lectura de un dato y lectura de múltiples datos.

B. Implementación con pines de GPIO (IIC virtual).

Para terminar de comprender el protocolo, se recomienda implementar un módulo IIC virtual, es decir, implementado usando solamente instrucciones para usar los pines como GPIO y alguna forma de generar los tiempos requeridos por el protocolo, sin usar el módulo de IIC que tiene el microcontrolador.

Lo anterior, además de reforzar el entendimiento del protocolo, prepara ante la situación de que los puertos de comunicaciones no sean suficientes y sea factible implementarlo vía software.

Se recomienda que el dispositivo IIC con el que se prueba el código sea una memoria, porque es el esquema más simple y donde fácilmente se puede extrapolar a otros dispositivos.

C. Configuración de reloj.

Es importante analizar la ecuación, que presenta el fabricante del microcontrolador, para definir la velocidad de comunicación. Ubicando las frecuencias estándar para IIC son 100 KHz y 400 KHz.

La ecuación para definir la velocidad de comunicación es la siguiente

$$UCABR1{:}UCABR0 = \frac{IIC_{Clock}}{Bit\ rate} = \frac{1\ MHz}{100\ Kbps}$$

Donde:

UCABR1:UCABR0 son los registros donde se guarda el valor para definir la velocidad de comunicación. UCABR1 es el byte alto y UCABR0 es el byte bajo.

IIC clock es el reloj seleccionado para ser la base de tiempo. En el ejemplo se selecciona SMCLK con los valores por *default.*

Bit rate es la velocidad de comunicación seleccionada.

D. Comunicación con sensor o actuador digital.

Es importante visualizar en un analizador lógico la trama de comunicación IIC, donde se ubiquen el bit de start, el identificador del esclavo, el *acknowledge*, la dirección a acceder y demás bits hasta llegar al bit de stop. Finalmente se recomienda desarrollar un código para interconectar un módulo IIC al microcontrolador utilizando el módulo interno IIC, de tal forma que sirva como código de referencia.

4.8. Comunicación SPI (*Serial Peripheral Interface*).

A. Estructura de la trama de comunicación.

El primer paso es encontrar los pines que pueden ser utilizados para la comunicación SPI y la manera de configurarlos.

El segundo paso es, a partir del diagrama de interconexión presente en el manual, definir la manera en que se conectan las señales entre el maestro y el (los) esclavo(s), y los módulos internos que permiten la comunicación especificada en el protocolo.

B. Configuración de reloj.

Se presenta el diagrama de los cuatro esquemas de generación del reloj, usando las variables de polaridad y fase. Es importante mencionar que dependiendo del dispositivo que se va a conectar al microcontrolador es necesario ajustar el esquema de reloj necesario. Por ejemplo, si el dispositivo tiene fijo tanto polaridad como fase, entonces la configuración en el microcontrolador debe tomar el esquema fijo. En contra parte, si el dispositivo con el que se establecerá la comunicación tiene polaridad y/o fase configurables, entonces tanto el microcontrolador como el dispositivo deben tener la misma configuración.

C. Comunicación con sensor o actuador digital.

Finalmente se recomienda desarrollar un código para interconectar un módulo SPI al microcontrolador utilizando el módulo interno SPI, de tal forma que sirva com código de referencia. Se sugiere usar un componente simple, como

pudiera ser un potenciómetro digital. De tal manera que la palabara digital que envía el dispositivo maestro, defina el valor de resistencia entre las terminales del dispositivo.

4.9. Entradas y salidas temporizadas.

A. Input Capture (IC). Medición de tiempos: anchos de pulso y frecuencia.

La recomendación es iniciar con un repaso y profundización de los modos de operación de los *timers*, revisando las banderas que reportan diferentes eventos en el conteo.

El concepto de IC o entrada temporizada, permite asociar el momento en que se presenta un evento en una entrada digital y el valor que tenía un temporizador en ese momento. Se considera interesante relacionarlo con manejos similares en la vida cotidiana como lo sería un reloj checador o el *time stamp* que se guarda cuando se reciben mensajes o llamadas telefónicas en un teléfono celular.

Para el caso de que la aplicación sea la medición de la frecuencia, es deseable evitar el uso de variables en formato punto flotante y calcular el recíproco. Se recomieda plantear la ecuación de la relación de periodo a frecuencia de otra manera distinta a la convencional, usando solo valores enteros.

B. Output Compare (OC). Generación de PWM, generación de ondas cuadradas ad-hoc. Motor DC.

Se sugiere iniciar el tema de OC o salida temporizada, utilizando la analogía del reloj despertador totalmente mecánico y después analizar la forma en que operan las alarmas de un smartphone. En donde se tiene un *timer* que constantemente se está comparando con diferentes referencias, que serían el equivalente a las alarmas.

Se recomienda generar PWM usando el modo *output compare*, ya que en algunos escenarios es conveniente hacerlo de esta manera además de que didáctamente fortalece el entendimiento del principio de operación del PWM.

C. Pulse Width Modulation (PWM). Edge Aligned/Center Aligned. Death band. Motor AC.

Se analizan las formas en que se generan los PWM, tanto en el formato *edge aligned* (*duty cycle* alineado a un extremo del periodo) o *center aligned* (*duty cycle* centrado en el periodo).

Para todo el tema, es importante hacer uso adecuado de las interrupciones. Considerar tiempos muy pequeños y muy grandes, ubicando como se puede complementar con software las capacidades del hardware cuando así se requiera.

Es importante el uso del analizador lógico para visualizar claramente los diagramas de tiempo y como se configuran las temporizaciones al cambiar los registros de configuración.

5. Evaluación de los cursos impartidos.

Es muy importante señalar que en el curso anterior se utilizaba un microcontrolador (8051) puesto en el mercado a inicios de los 80's, por lo que actualmente se considera una arquitectura computacional elemental que no permitía tener mayores capacidades de procesamiento. Por otro lado, el curso se impartía completamente en lenguaje ensamblador, por lo que hacía muy lento el proceso de desarrollo de una aplicación.

Después de haberse impartido el nuevo curso en tres ocasiones, se presentan a continuación los elementos que permiten evaluar los resultados de la propuesta del nuevo curso. En general se presentarán los resultados en dos planos. Uno asociado a la reflexión del profesor y otro utilizando elementos cuantitativos provenientes del IAE (Instrumento de Apreciación Estudiantil).

En las figuras 9 a 12 se resumen las respuestas para las preguntas 2 a la 16. Esta es la manera en que los instrumentos de apreciación estudiantil reportan la información.

Revisando la gráfica comparativa mostrada en la Figura 9, entre la UAB-E y la primera ocasión en que se imparte el curso para el Profesor 1, solo en la pregunta 4 ("Han promovido que aprenda a partir de mis propias acciones") el curso tiene menor valoración que la UAB-E.

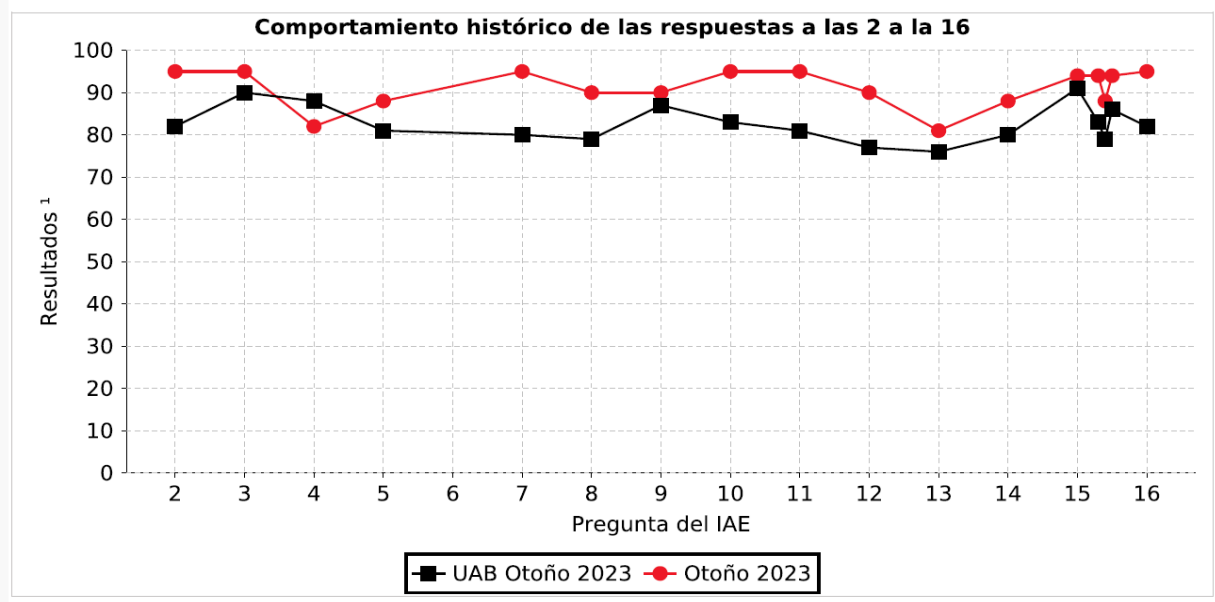

Figura 9 Instrumento de apreciación estudiantil otoño 2023 (Profesor 1)

De igual manera las Figuras 10, 11 y 12 muestran una visión general del IAE para los semestres Primavera 2024 y dos grupos del semestre Otoño 2024.

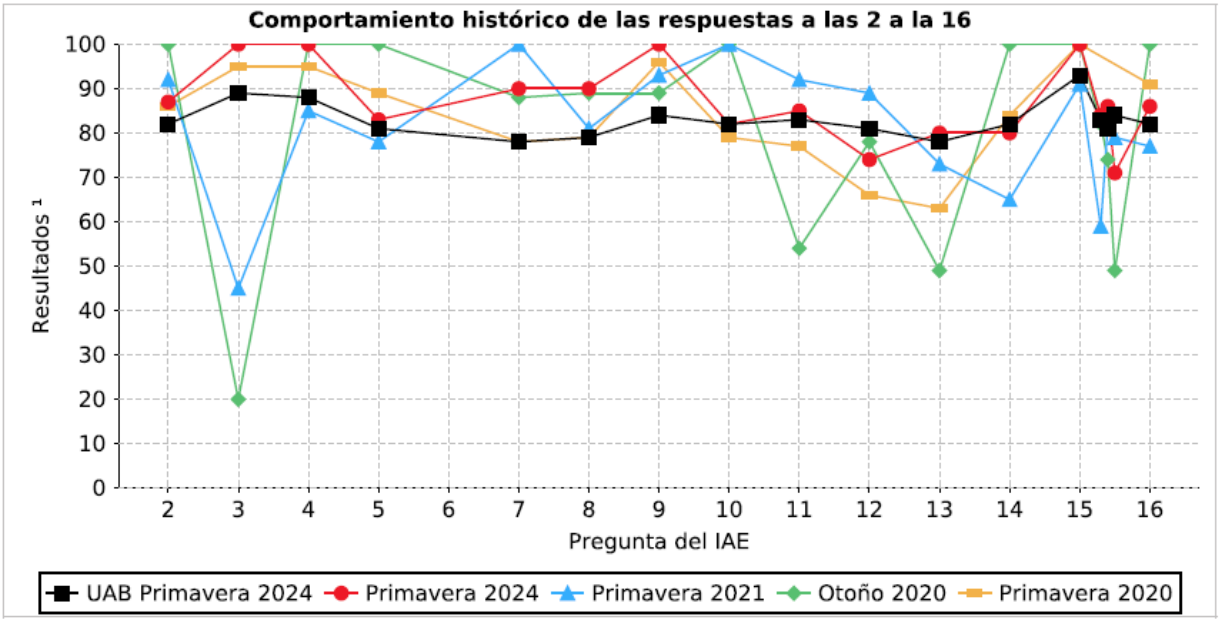

Figura 10 Instrumento de apreciación estudiantil primavera 2024 (Profesor 1)

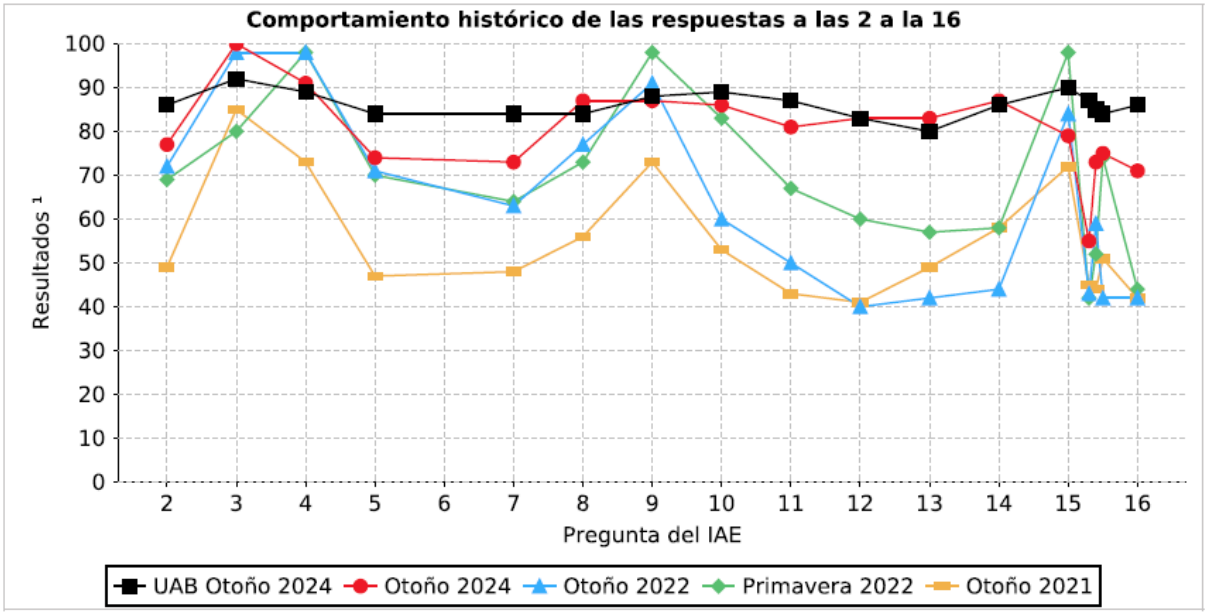

Figura 11 Instrumento de apreciación estudiantil otoño 2024 (Profesor 2)

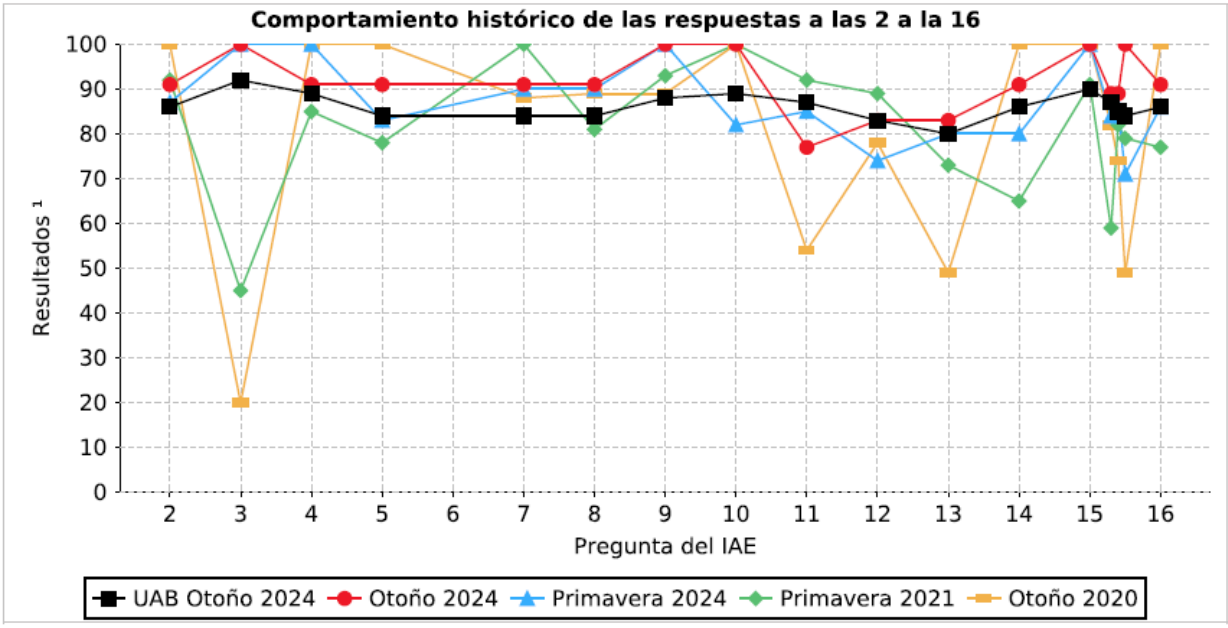

Figura 12 Instrumento de apreciación estudiantil otoño 2024 (Profesor 1)

Es importante aclarar que cualquier comparativo con cursos antes de otoño 2023 no aplica, ya que es el curso con contenidos antes de las modificaciones planteadas en este documento. Este curso fue recibido por parte de los estudiantes con bastante buena motivación, desde el inicio entendieron que se trabajaría con un microcontrolador más reciente, que permitía hacer aplicaciones más interesantes, un dispositivo con mayores capacidades y con un trabajo más enfocado en la programación y aplicaciones que en el alambrado de memorias externas.

Para la pregunta 17, "¿Cuál es tu nivel de satisfacción con los aprendizajes que has obtenido hasta el momento en esta asignatura?", las Figuras 13, 14, 15 y 16 presentan las valoraciones, respectivamente de otoño 2023, primavera 2024 y dos grupos de otoño 2024. En todas las evaluaciones al menos el 70% evalúan la satisfacción entre alta y muy alta.

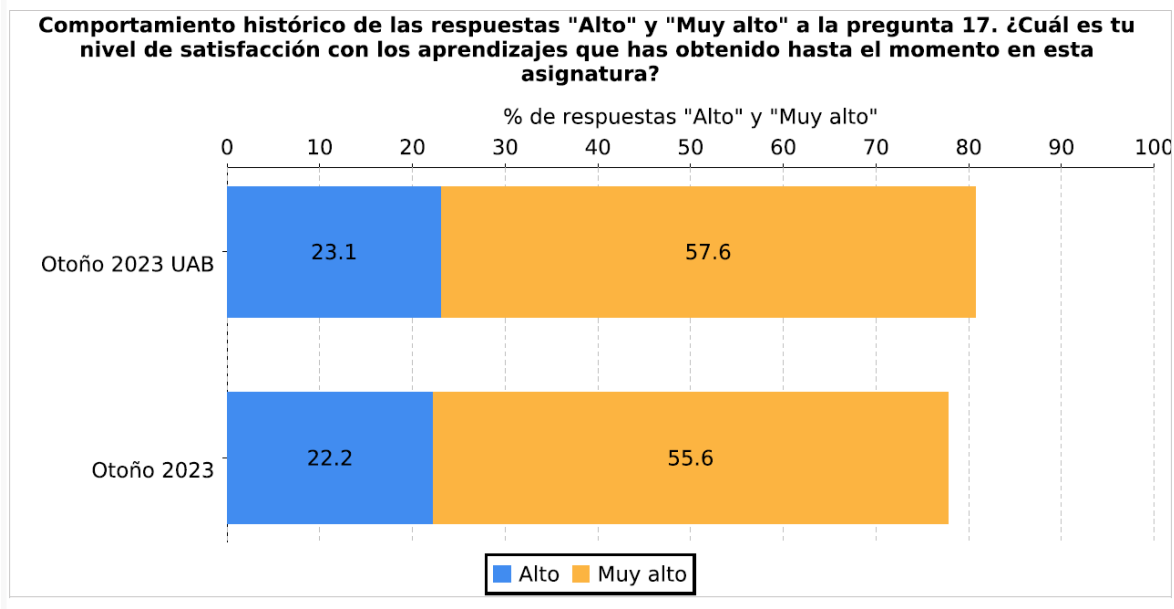

Figura 13 IAE pregunta 17 Otoño 2023 (Profesor 1)

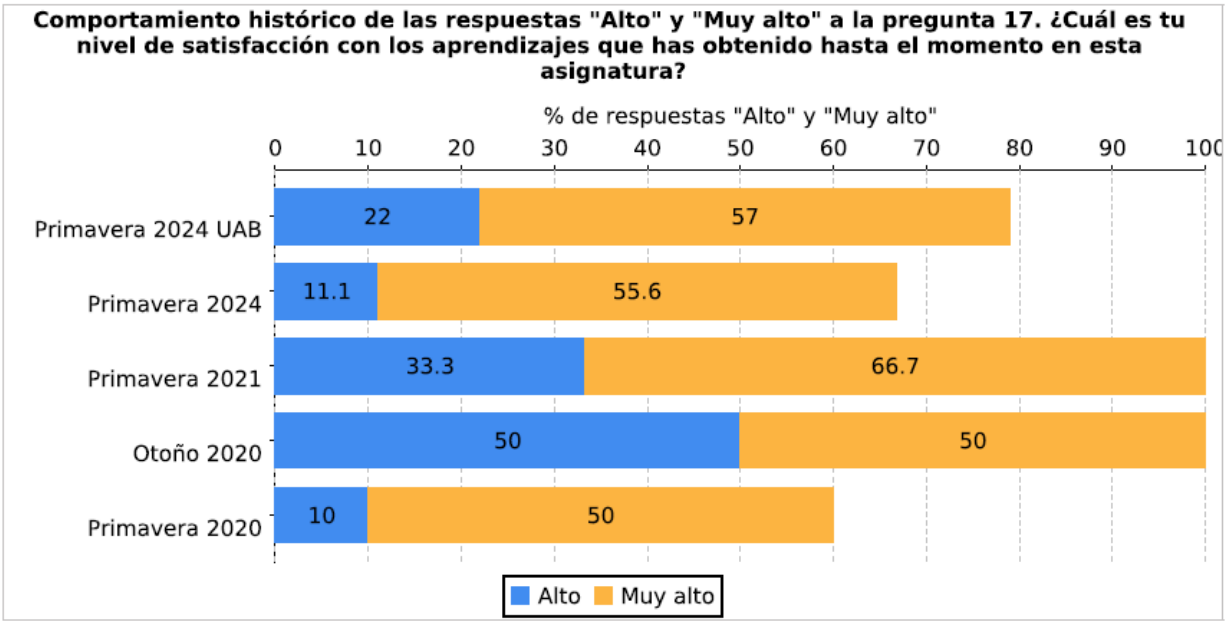

Figura 14 IAE pregunta 17 Primavera 2024 (Profesor 1)

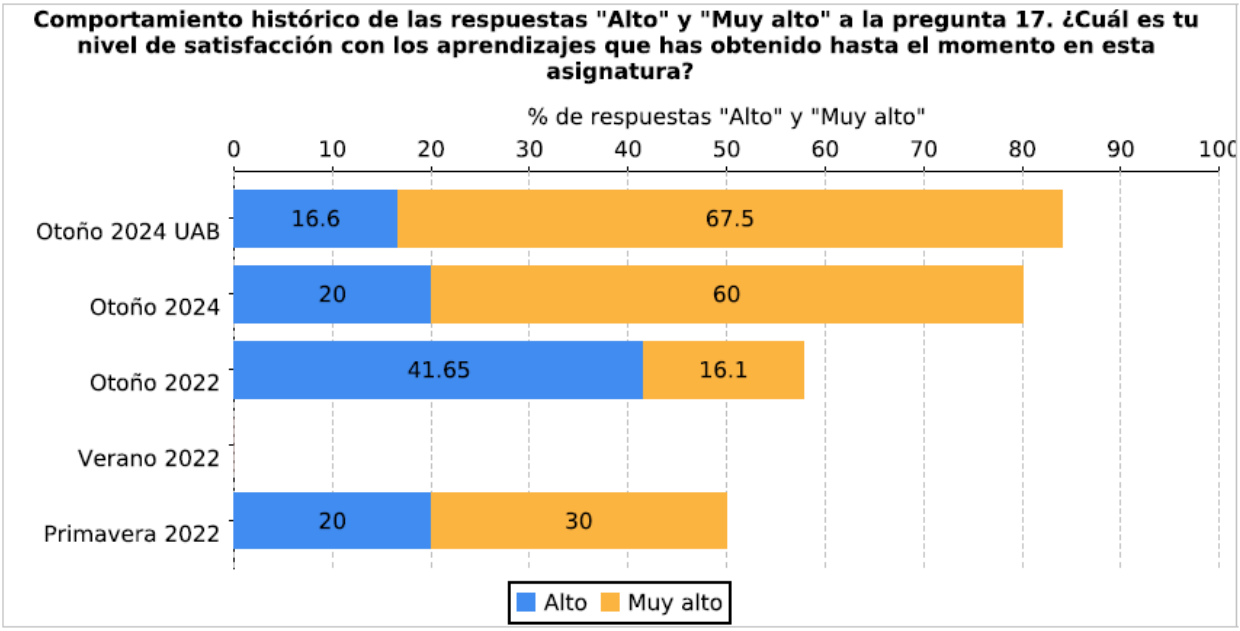

Figura 15 IAE Pregunta 17 Otoño 2024 (Profesor 2)

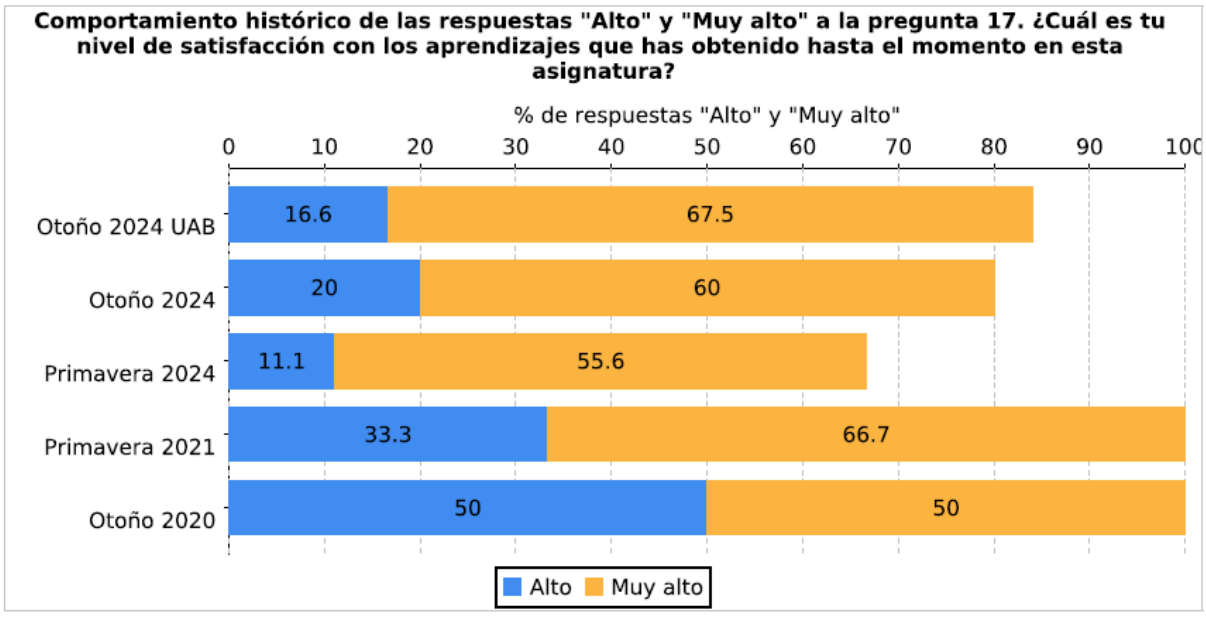

Figura 16 IAE Pregunta 17 Otoño 2024 (Profesor 1)

En el caso de la pregunta 18, donde se cuestiona sobre cambios al curso, a continuación, los resultados. Las figuras 17, 18, 19 y 20, respectivamente, para los semestres otoño 2023,

primavera 2024 y los dos grupos de otoño 2024. En tres de los cuatro cursos la mayoría de los estudiantes no propone cambios para el curso.

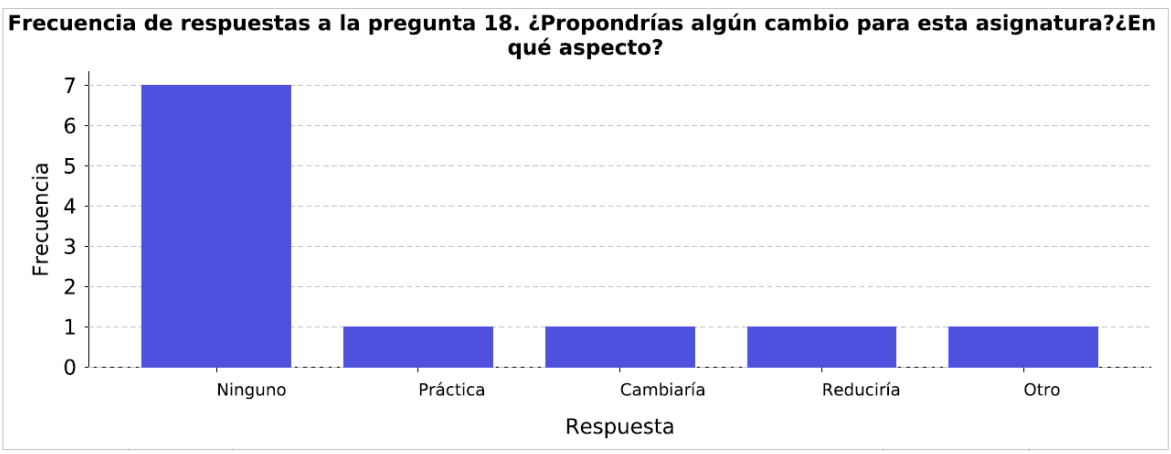

Figura 17 IAE pregunta 18 otoño 2023 (Profesor 1)

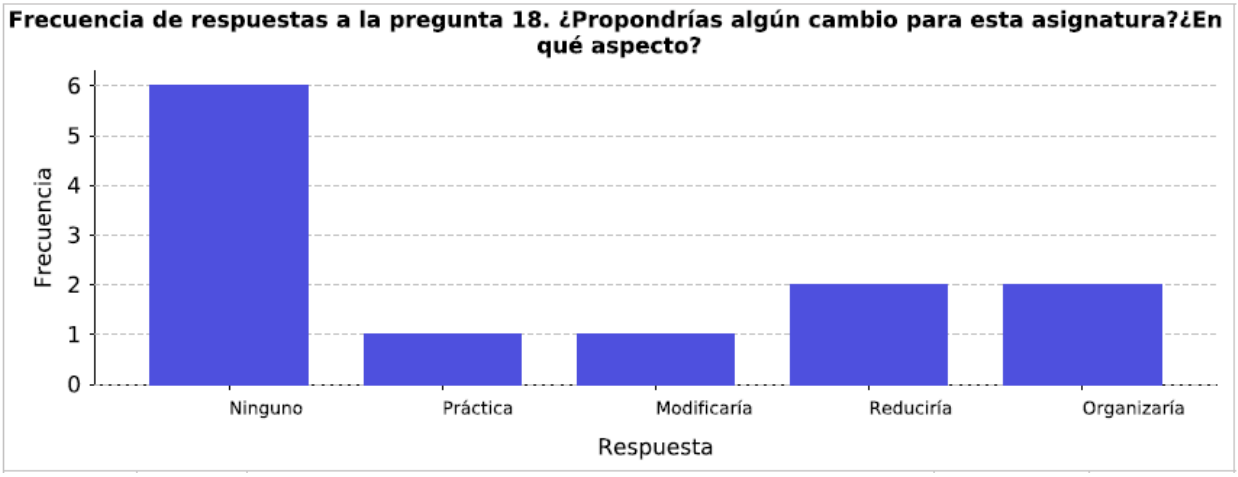

Figura 18 IAE pregunta 18 Primavera 2024 (Profesor 1l)

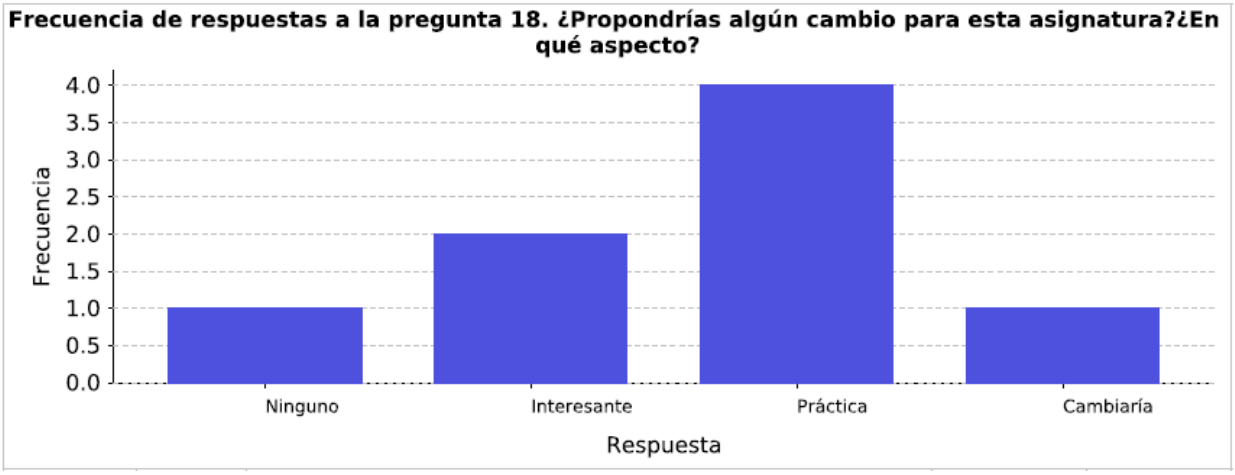

Figura 19 IAE pregunta 18 Otoño 2024 (Profesor2)

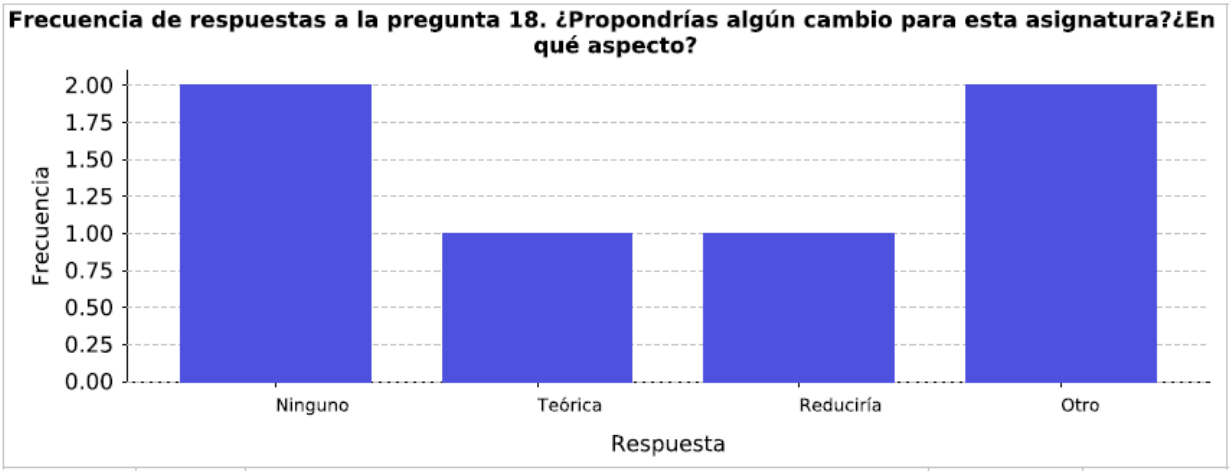

Figura 20 IAE pregunta 18 Otoño 2024 (Profesor 1)

5.1. Sesiones bajo conducción docente.

Respecto al calendario original, las sesiones se dieron en general de manera adelantada, es decir, varios temas se cubrieron en menos tiempo. Lo que permitió tener algunas sesiones para utilizarlas como taller de prácticas o proyecto final.

De cualquier manera, se recomienda mantener el plan de sesiones originales y dejar los tiempos restantes, si los hubiera, para dar asesoría a los estudiantes. En ningún momento se percibió de parte del profesor o de los estudiantes que se fuera a un ritmo muy pesado. El tiempo

asignado exigía al estudiante estar atento, hacer sus actividades y permanecer concentrado en las sesiones.

Se realizó un ajuste, donde justo después de estudiar la interfaz serial asíncrona, se revisó el tema de entradas y salidas temporizadas, lo anterior para que tuvieran claridad del manejo de motores lo más pronto posible. Este cambio es dependiente del tipo de proyecto, por lo que, si cambia dicha actividad, podrían reacomodarse los temas del curso.

5.2. Trabajo del estudiante: Ejercicios.

Se considera que la metodología de desarrollar una cantidad de ejercicios considerable, nueve en total, ayudó a que cada tema se fuera reforzando con el trabajo práctico. Aunque al principio el estudiante estaba un tanto abrumado con un promedio de un ejercicio por semana. Al terminar el curso, los estudiantes hacen referencia a ejercicios previos para validar sus prácticas y al final para el proyecto. Lo que les servía como proceso de depuración, primero para validar el hardware y el funcionamiento aislado, antes de integrarlo en un problema de mayor complejidad.

El ejercicio de SPI se integró al proyecto final. Se considera que debe de hacerse un ejercicio básico con SPI, conectando un módulo mucho más simple que el módulo de radio.

El ejercicio de IIC no tuvo los resultados esperados. Es importante dar un poco más de guía a los estudiantes.

Se estimaron los tiempos para que el tiempo independiente del estudiante (TIE) se usara en el desarrollo de los ejercicios y prácticas y en repasar las sesiones de clase.

5.3. Trabajo del estudiante: Prácticas de laboratorio.

Las primeras dos prácticas de laboratorio cumplieron con los objeticos planteados. En la primera práctica se integraron elementos de *timers* y ADC, ambos vía interrupciones, así como el manejo de un motor que actúa sobre la transmisión del vehículo del proyecto y el otro a la dirección del carro. Los estudiantes elaboraron sus mecanismos equivalentes al volante y al pedal de acelerador del proyecto final.

Para la segunda práctica incorporaron el manejo de un módulo de comunicaciones inalámbrico, interfaz para una aplicación en la computadora y la decodificación de la trama generada por un GPS. Los resultados fueron los esperados lo que permitió que los estudiantes avanzaran en entregables asociados también al proyecto final.

Para la tercera práctica, se tuvo un problema en el manejo de tiempo y se prefirió integrar la práctica al entregable del proyecto. El principal problema fue el módulo de radio frecuencia para alcanzar distancias mayores. Es un módulo complejo para que los estudiantes desarrollen el código desde cero, por lo que se buscaba que integraran código que encontraran en internet.

5.4. Trabajo del estudiante: Exámenes

Los exámenes incluyen preguntas teóricas presentadas como de respuesta abierta. Lo anterior busca desarrollar la capacidad de expresarse correctamente ante una descripción técnica. No se permite consultar ningún tipo de información. El punto central es el manejo de los conceptos y que puedan describirse correctamente.

Además de las preguntas teóricas se incluye al menos una pregunta donde el estudiante debe de diseñar una combinación hardware y software que resuelva un problema específico. Para la pregunta práctica pueden auxiliarse de códigos desarrollados en clase o en sus prácticas y ejercicios, así como cualquier documentación de los manuales del microcontrolador y de la tarjeta de desarrollo. Por lo que, si sabe integrar código anteriormente desarrollado, puede tener una solución en un tiempo acotado.

En cursos similares, es común que el primer examen tenga resultados no muy buenos para los estudiantes. Lo anterior sucedió en el curso, pero los estudiantes fueron conscientes de sus errores y trabajaron en ellos. Desde el punto de vista del profesor, el resultado del primer examen no evidenciaba falta de aprendizajes, sino falta de pericia para poder integrarlos en un tiempo acotado.

Mucha mejoría en los resultados del segundo examen, donde ya desarrollaron la capacidad de diseñar una solución hardware-software, para cumplir con un requerimiento en un tiempo acotado.

5.5. Trabajo del estudiante: Proyecto.

Aunque el proyecto final se fue desarrollando con las prácticas, durante todo el semestre, al final implicó un esfuerzo importante de los estudiantes. En el Apéndice D se presenta el enunciado del proyecto final.

El proyecto integra un porcentaje considerable de las competencias desarrolladas durante el semestre en un ambiente práctico. Para muchos estudiantes el desarrollo del proyecto implicó reforzar conocimientos y habilidades o construirlas para cumplir con los entregables del proyecto.

Aunque es un curso de fundamentos, se considera importante tener un proyecto integrador, todos los estudiantes haciendo el mismo proyecto, sin la necesidad de que ellos lo propongan. En este curso se plantea el desarrollo del proyecto más como una justificación concreta de usar muchas competencias en un solo entregable.

En el curso piloto, no se cumplieron totalmente los objetivos del proyecto. A cada equipo le faltó al menos un elemento, todos los subsistemas fueron puesto a funcionar al menos por un equipo.

Poner a funcionar un sistema con baterías también presentó un reto, donde los estudiantes tenían que definir el tipo de baterías, la cantidad de voltaje, la demanda de corriente y donde los estudiantes debían montarlas en el ensamble final.

5.6. Valoración de profesor invitado.

Para el curso de primavera 2024 se decidió que un profesor se integrara al curso como escenario de capacitación sobre las cuestiones técnicas y metodológicas del curso, de tal manera que pudiera ser profesor en futuros cursos.

A continuación, se integran algunos de los comentarios presentados por el profesor. El texto es sin modificaciones para respetar la forma y fondo original:

Después de asistir a la mayoría de las sesiones del curso, **estas son mis impresiones**:

i) El material presenta una curva de aprendizaje pronunciada al emplear muchas variables y constantes propias del microcontrolador.
ii) Los manuales, como se comentó en clase, no son lo más adecuado pedagógicamente hablando. Es cierto que en la industria quizá sólo se cuente con ese acervo, pero para ser un curso introductorio, quizá sería bueno contar con un poco más de material de apoyo.
iii) La tarjeta resuelve en buena parte problemas de hardware que quedan sin exponerse a los alumnos o al menos no se preocupan al ya tenerlos resueltos.
iv) Dedicar un poco de tiempo a la arquitectura y al IDE. Conocer un poco más los mapas de memoria, empezar a conocer algunos registros, como el PC o el SP que ayuden a entender procesos como las interrupciones, localización de variables locales, globales, etc. Respecto al IDE, aunque en cada unidad se pueden ver apartados relacionados con esos tópicos, dar una visión general creo que sería una aproximación más amable.

Por ello sugiero:

i) Contar con un banco de ejercicios sencillos donde se puedan practicar más la configuración de los distintos elementos del *micro* y se haga familiar el uso de nombres, constantes, registros.
ii) Adquirir para biblioteca un par de libros, por ejemplo [20] y [21].
iii) Valorar al menos una práctica donde se programe el micro y se pase a tarjeta tipo *protoboard* para desde ahí enfrentar al alumno con aspectos básicos de hardware.

Dichos comentarios se están revisando para evaluar la forma en ser considerados para futuros cursos.

Conclusiones

Este documento presenta la metodología seguida para la formulación del curso de Fundamentos de Microprocesadores y Microcontroladores: generalidades del curso, definición y desarrollo de los contenidos, material didáctico y actividades de aprendizaje. Se actualizó el curso a partir del análisis y la consideración de diversos elementos antecedentes, así como el estado actual de la tecnología. Se analizaron los elementos que nutren las necesidades del curso: variantes del mismo curso a través del tiempo, cursos similares en otras universidades, cursos parecidos para varias carreras que lo demandan y escenarios laborales profesionales donde se requieran estas competencias.

Se elaboró la guía de aprendizaje como primer nivel de especificación del curso, y se diseñaron las actividades de aprendizaje que desarrollará el estudiante: ejercicios, prácticas, proyecto final y exámenes.

Se elaboró el material didáctico y programas de referencia para el microcontrolador.

Hasta el momento en que se escribe este documento, se ha ofrecido el curso en dos ocasiones y se está ofertando por tercera ocasión.

Se considera que toda la información generada es valiosa para los profesores que imparten o impartirán este curso.

La metodología desarrollada permite poder actualizar el curso cuando sea necesario, por cambio tecnológico o por disponibilidad y costo del microcontrolador seleccionado.

Se sugiere que para los cursos subsecuentes en este tipo de competencias se siga una metodología similar. Mezclando elementos de la realidad del uso industrial de los microprocesadores y microcontroladores, con elementos metodológicos para la enseñanza y la experiencia de los profesores en el proceso de aprendizaje.

Otra sugerencia para ser considerada es cambiar el nombre del curso a Programación e interconexión de Microcontroladores. Un nombre que indique, con un poco más de detalle, las competencias que desarrolla el estudiante.

Quedan pendientes dos elementos muy importantes para evaluar las modificaciones a este curso: El primero es cómo lo evalúan los profesores de otros cursos en donde se utilizan las

competencias desarrolladas en este curso, y el segundo elemento es la evaluación del estudiante, una vez que requiera estas competencias para utilizarlas en su vida profesional. Esta retroalimentación en años futuros será muy valiosa para el proceso de mejora continua que requiere un curso y, por ende, los programas de estudio de las ingenierías que lo incorporan.

Se evaluará el uso de repositorios para guardar los códigos las actividades de los estudiantes. Al ser un curso con tantos aprendizajes, existe el riesgo que sea un distractor. Por otro lado, los códigos son muy simples y no reflejan el caso de uso industrial, por lo que pueda generar una percepción errónea del uso.

Bibliografía

[1] B.B. Brey. *The Intel Microprocessors*, 8th ed. Prentice Hall Press, 2009.

[2] Cai Limin, "Thought on embedded system education for application-oriented undergraduates of electronics major," *2009 4th International Conference on Computer Science & Education*, Nanning, China, 2009, pp. 1476-1478, doi: 10.1109/ICCSE.2009.5228568.

[3] H. Fan, X. Wu, R. Ghannam, Q. Feng, H. Heidari and M. A. Imran, "Teaching Embedded Systems for Energy Harvesting Applications: A Comparison of Teaching Methods Adopted in UESTC and KTH," in IEEE Access, vol. 8, pp. 50780-50791, 2020, doi: 10.1109/ACCESS.2020.2980336.

[4] NSK Surgic Pro (2025, Feb. 24). Motor para implante dental NSK Surgic Pro. Available: https://www.latin-america.nsk-dental.com/admin/wp-content/uploads/PR-D1421ESv1_SURGICAL_241225.pdf

[5] Geomax (2025, Feb. 24). Estación total Zoo 25 Pro A5 de 1 segundo. Available: https://estaciontotal.info/wp-content/uploads/Estacion-total-Geomax-zoom-25.pdf

[6] Omega. (2025, Feb. 24). Registrador de temperatura y humedad HOBO MX Bluetooth con pantalla. Available: https://www.onsetcomp.com/sites/default/files/2023-01/17840-U%20MX1101%20Manual.pdf?srsltid=AfmBOorM_Y81zxrwSGfkxHdE7oDe4JtFBUkDdwFgdMUjS4164c--psAM

[7] IJRASET. (2025, Feb 24). Arduino working principle and it's use in education. Available: https://www.ijraset.com/research-paper/arduino-working-principle-and-its-use-in-education

[8] O.A. Patiño et al. "Evolution of Microcontroller's Course under the influence of Arduino," *LACCEI International Multi-Conference for Engineering, Education, and Technology*. San Jose, Costa Rica. July 2016, doi: http://dx.doi.org/10.18687/LACCEI2016.1.1.183

[9] All about circuits. (2025, Feb 24). How to read a microcontroller datasheet: Introduction and first steps. Available:https://www.allaboutcircuits.com/technical-articles/how-to-read-a-microcontroller-datasheet-introduction-and-first-steps2/

[10] All about circuits. (2025, Feb 24). How to choose the right microcontroller for your application. Available: https://www.allaboutcircuits.com/technical-articles/how-to-choose-the-right-microcontroller-for-your-application/

[11] H. Kobayashi, K. Kumamoto, T. Yoshimura, T. Haga, M. Iyota and M. Katoh, "A Design of Engineering PBL on Embedded System for Novice Freshmen Students," *TENCON 2018 - 2018 IEEE Region 10 Conference*, Jeju, Korea (South), 2018, pp. 0782-0786, doi: 10.1109/TENCON.2018.8650212.

[12] Texas Instruments. (2025, Feb. 24). MSP-EXP430G2ET. Available: https://www.ti.com/tool/MSP-EXP430G2ET

[13] N. K. Uttarkar and R. R. Kanchi, "Design and development of a low-cost embedded system laboratory using TI MSP430F149," *2013 International Conference on Communication and Signal Processing*, Melmaruvathur, India, 2013, pp. 165-175, doi: 10.1109/iccsp.2013.6577037.

[14] Z. Jusoh, R. Darus, M. T. Miskon and S. Omar, "Hands-on and training on embedded system," *2015 IEEE 7th International Conference on Engineering Education (ICEED)*, Kanazawa, Japan, 2015, pp. 151-154, doi: 10.1109/ICEED.2015.7451510.

[15] A. Hunter. (2025, Feb. 24). PIC32 Microcontroller Lectures. Designing with Microcontrollers course. Available: https://www.youtube.com/playlist?list=PLDqMkB5cbBA6FEJuj94gl-9vw8xcHu9Gp

[16] L. Xiaojuan, G. Yong and Y. Huimei, "Curriculum Development and Progressive Engineering Practice Design in Embed System Education," *2008 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*, Beijing, China, 2008, pp. 228-232, doi: 10.1109/MESA.2008.4735752.

[17] Z. Jusoh, H. Husni, S. I. Ismail, S. Omar and R. Abdullah, "Implementation of embedded system design in student's final year project using problem based learning approach," *2017 IEEE 9th International Conference on Engineering Education (ICEED)*, Kanazawa, Japan, 2017, pp. 201-205, doi: 10.1109/ICEED.2017.8251193.

[18] A. F. Mondragon-Torres and J. W. Christman, "A comprehensive embedded systems design course and laboratory," *2013 IEEE International Conference on Microelectronic Systems Education (MSE)*, Austin, TX, USA, 2013, pp. 56-59, doi: 10.1109/MSE.2013.6566704.

[19] Naylamp. (2025, Feb. 24). Tutorial básico de uso del módulo bluetooth HC’06 y HC05. Available: https://naylampmechatronics.com/blog/12_tutorial-basico-de-uso-del-modulo-bluetooth-hc-06-y-hc-05.html

[20] J.H. Davies, *MSP430Microcontroller Basics*, Newnes Elsevier. 2008

[21] M. Jimenez et al. *Introduction to Embedded Systems Using Microcontrollers and the MSP430*, Springer. 2014

Apéndices

Si el lector requiere más detalle del material de la clase, por favor contacte al autor: msinseld@iteso.mx

A. EXPERIENCIA PROFESIONAL Y ACADEMICA DEL AUTOR

La vida profesional del autor cuenta con más de 35 años de experiencia como profesor universitario y como profesionista en el campo de la Ingeniería Electrónica.

En este periodo de tiempo se cuentan con un número importante de estudiantes en cursos de microcontroladores, se estima en alrededor de 2500 estudiantes. Los cursos impartidos incluyen cursos para estudiantes de las carreras de Ingeniería Electrónica, Ingeniería Mecatrónica, Ingeniería Biomédica e Ingeniería en Sistemas Computacionales. Los cursos fueron impartidos en el Instituto Tecnológico y de Estudios Superiores de Occidente (ITESO) y en el Instituto Tecnológico y de Estudios Superiores de Monterrey Campus Guadalajara.

Además de los cursos de Licenciatura, también se han impartido cursos sobre Microcontroladores en programas de Posgrado: ITESO, Tec de Monterrey y CINVESTAV unidad Guadalajara.

En otros escenarios, se participó como instructor certificado de Texas Instruments y Motorola Semiconductores en cursos para profesores, investigadores e ingenieros de la industria. Los cursos estaban enfocados en Microcontroladores y DSPs (Procesadores Digitales de Señales).

Por otro lado, en el campo industrial, se han puesto en práctica los conocimientos y experiencias en el campo en empresas como: Arquitectura de Sistemas Computacionales Integrales (ASCI), Soluciones Tecnológicas, Idear Electrónica (Sistemas BEA), Pegasus Control, Freescale, Continental Automotive Guadalajara, NXP y BOSCH México. Además de muchos otros proyectos como Freelance.

El rol como ingeniero de sistemas embebidos incluyen: Ingeniero de diseño y desarrollo, *Team Leader*, *Technical Leader*, *Principal Staff Engineer*, Director de Ingeniería, Gerente de Desarrollo de Competencias.

Como tema adicional importante, se desarrolló la primera propuesta de la Especialidad de Sistemas Embebidos del ITESO, además de participar en el diseño de diferentes materias para los programas de estudio tanto del ITESO como del Tec de Monterrey. Mención aparte la colaboración en el diseño de la Maestría en Sistemas Embebidos desarrollada por CONACYT.

También se tuvo una colaboración como coautor de un libro titulado “Microcontroladores Motorola – Freescale” que en algún tiempo fue parte la bibliografía de un curso en la Universidad de Guadalajara.

Se tuvo la circunstancia de ser profesor de posgrado de docentes en el área de Microcontroladores de la Universidad Autónoma de Guadalajara y del Centro de Enseñanza Técnica e Industrial plantel Colomos. En diversos diálogos se influyó en los contenidos y metodologías que se han usado en dichas instituciones de educación superior

Todo lo anterior se considera como una mezcla valiosa entre experiencia docente y profesional que propone las competencias a desarrollar y la sugerencia de metodología.

B. ENUNCIADOS DE LOS EJERCICIOS

Ejercicio 1. Instalar IDE y probar *tool chain*
Ingeniería en Mecatrónica (O2023)

Objetivo:
Puesta en marcha del ambiente IDE (Integrated Development Environment) del microcontrolador a utilizar en el curso

Enunciado:
A partir de un código básico. Realizar las siguientes actividades:

- Crear un proyecto
- Insertar el código de referencia
- Compilar el programa
- Transferir el código al microcontrolador
- Ejecutar paso a paso para verificar la funcionalidad del código

Materiales:

- Tarjeta TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Cable USB
- Code Composer Studio – IDE (CCS)

Entregable:
Se debe de subir a CANVAS un video donde se realicen las actividades descritas en el enunciado. El profesor puede revisar de manera presencial las actividades.
El trabajo es en equipo de dos personas.

Programa básico:

```
#include <msp430.h>

unsigned char i;

int main(void)
{
	WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
	for(;;)
	{
	    i++;
	}
	return 0;
}
```

Ejercicio 2. Blinking LED
Ingeniería en Mecatrónica (O2023)

Objetivo:
Puesta en operación del primer código con pines en modo GPIO

Enunciado:
A partir del código realizado en clase. Realizar las siguientes actividades:

- Crear un proyecto
- Insertar el código realizado en clase
- Compilar el programa
- Transferir el código al microcontrolador
- Ejecutar paso a paso para verificar la funcionalidad del código

Materiales:

- Tarjeta TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Cable USB
- Code Composer Studio – IDE (CCS)

Entregable:
Se debe de subir a CANVAS el código a probar, agregarlo en un repositorio remoto y un video donde se realicen las actividades descritas en el enunciado. El profesor puede revisar de manera presencial las actividades.

El trabajo es en equipo de dos personas.

Ejercicio 3. Stepper motor 1
Ingeniería en Mecatrónica (O2023)

Objetivo:
Puesta en operación del primer código para manejo de un motor a pasos

Enunciado:
A partir del código realizado en clase. Realizar las siguientes actividades:

- Crear un proyecto
- Ajustar el código generado en clase para las siguientes funcionalidades
    - Movimiento del motor con dos fases activas
    - La dirección del sentido del giro del motor será controlado por el botón de usuario que se encuentra disponible en la tarjeta electrónica.
    - Se puede cambiar la velocidad de giro del motor.
- Verificar la funcionalidad.

Materiales:

- Tarjeta TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Cable USB
- Code Composer Studio – IDE (CCS)
- Driver para el manejo del motor a pasos. NO CONECTAR DIRECTAMENTE EL MOTOR A LOS PINES DEL MICROCONTROLADOR.
- Motor a pasos

Entregable:
Se debe de subir a CANVAS el código a probar, agregarlo en un repositorio remoto y un video donde se realicen las actividades descritas en el enunciado. El profesor puede revisar de manera presencial las actividades.

El trabajo es en equipo de dos personas.

Ejercicio 4. Stepper motor 2
Ingeniería en Mecatrónica (O2023)

Objetivo:
Puesta en operación del código para manejo de un motor a pasos usando *timer* para definir la velocidad

Enunciado:
A partir del código realizado en clase. Realizar las siguientes actividades:

- Crear un proyecto
- Ajustar el código generado en clase para las siguientes funcionalidades
    - Movimiento del motor con dos fases activas
    - La dirección del sentido del giro del motor será controlada por el botón de usuario que se encuentra disponible en la tarjeta electrónica.
    - Se puede cambiar la velocidad de giro del motor usando un *timer*
- Verificar la funcionalidad.

Materiales:

- Tarjeta TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Cable USB
- Code Composer Studio – IDE (CCS)
- Driver para el manejo del motor a pasos. NO CONECTAR DIRECTAMENTE EL MOTOR A LOS PINES DEL MICROCONTROLADOR.
- Motor a pasos

Entregable:
Se debe de subir a CANVAS el código a probar, agregarlo en un repositorio remoto y un video donde se realicen las actividades descritas en el enunciado. El profesor puede revisar de manera presencial las actividades.

El trabajo es en equipo de dos personas.

Ejercicio 5. Stepper motor 3. Interrupciones
Ingeniería en Mecatrónica (O2023)

Objetivo:
Puesta en operación del código para manejo de un motor a pasos usando *timer* para definir la velocidad y usando la ISR del *timer* para actualizar el paso.

Enunciado:
A partir del código realizado en clase. Realizar las siguientes actividades:

- Crear un proyecto
- Ajustar el código generado en clase para las siguientes funcionalidades
    - Movimiento del motor con dos fases activas
    - La dirección del sentido del giro del motor será controlada por el botón de usuario que se encuentra disponible en la tarjeta electrónica.
    - Se puede cambiar la velocidad de giro del motor usando un *timer*.
    - La secuencia de valores se genera en la ISR del *Timer*
- Verificar la funcionalidad.

Materiales:

- Tarjeta TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Cable USB
- Code Composer Studio – IDE (CCS)
- Driver para el manejo del motor a pasos. NO CONECTAR DIRECTAMENTE EL MOTOR A LOS PINES DEL MICROCONTROLADOR.
- Motor a pasos

Entregable:
Se debe de subir a CANVAS el código a probar, agregarlo en un repositorio remoto y un video donde se realicen las actividades descritas en el enunciado. El profesor puede revisar de manera presencial las actividades.

El trabajo es en equipo de dos personas.

Ejercicio 6. Movimiento del motor a pasos controlado por un volante (ADC)
Ingeniería en Mecatrónica (O2023)

Objetivo:
Puesta en operación del código para manejo de un motor a pasos usando un potenciómetro (ADC) para definir la posición.

Enunciado:
A partir del código realizado en clase. Realizar las siguientes actividades:

- Crear un proyecto
- Ajustar el código generado en clase para las siguientes funcionalidades
    - Movimiento del motor con dos fases activas
    - La posición del motor estará controlada por la entrada de un potenciómetro (ADC)
    - Se debe de iniciar una conversión del ADC cada 10 ms vía la interrupción de un *timer*.
    - El ADC también debe de manejarse por interrupción.
- Verificar la funcionalidad.

Materiales:

- Tarjeta TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Cable USB
- Code Composer Studio – IDE (CCS)
- Driver para el manejo del motor a pasos. NO CONECTAR DIRECTAMENTE EL MOTOR A LOS PINES DEL MICROCONTROLADOR.
- Motor a pasos
- Potenciómetro

Entregable:
Se debe de subir a CANVAS el código a probar, agregarlo en un repositorio remoto y un video donde se realicen las actividades descritas en el enunciado. El profesor puede revisar de manera presencial las actividades.

El trabajo es en equipo de dos personas.

Ejercicio 7. Envío de valores de sensores vía UART
Ingeniería en Mecatrónica (O2023)

Objetivo:
Puesta en operación del código para leer dos entradas analógicas y enviarlas vía UART para ser visualizadas en una computadora.

Enunciado:
A partir del código realizado en clase. Realizar las siguientes actividades:

- Crear un proyecto
- Ajustar el código generado en clase para las siguientes funcionalidades
    - Se leerán, usando el ADC, dos entradas analógicas pudiendo provenir los valores de dos potenciómetros.
    - Se visualizarán los datos en una computadora
    - La frecuencia de muestreo se dará vía interrupción de un *timer*
    - La lectura de las entradas analógicas se hará vía interrupción del ADC.
    - La transmisión de datos de la UART se hará vía interrupciones.
    - Los datos deberán ser visualizados usando Matlab o Python
- Verificar la funcionalidad.

Materiales:

- Tarjeta TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Cable USB
- Code Composer Studio – IDE (CCS)
- Potenciómetro

Entregable:
Se debe de subir a CANVAS el código a probar, agregarlo en un repositorio remoto y un video donde se realicen las actividades descritas en el enunciado. El profesor puede revisar de manera presencial las actividades.

El trabajo es en equipo de dos personas.

Ejercicio 8. Control de velocidad de dos motores de CD vía UART
Ingeniería en Mecatrónica (O2023)

Objetivo:
Puesta en operación del código para leer valores de duty cycle y así controlar la velocidad y dirección de dos motores de corriente directa.

Enunciado:
A partir del código realizado en clase. Realizar las siguientes actividades:

- Crear un proyecto
- Ajustar el código generado en clase para las siguientes funcionalidades
    - Se leerá del puerto serial los valores de duty cycle para la generación de dos PWM. Así como la dirección de sentido de giro. Los datos se deben de leer vía interrupción del microcontrolador.
    - Las salidas de PWM se conectarán a la etapa de potencia necesaria para poder manejar los motores de corriente directa.
    - No debe de conectarse el motor directamente a los pines del microcontrolador.
    - Los datos deben de enviarse usando Matlab o Python
- Verificar la funcionalidad.

Materiales:

- Tarjeta TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Cable USB
- Code Composer Studio – IDE (CCS)
- Puentes H para manejar dos motores de corriente directa de manera bidireccional.
- Dos motores de corriente directa.

Entregable:
Se debe de subir a CANVAS el código a probar, agregarlo en un repositorio remoto y un video donde se realicen las actividades descritas en el enunciado. El profesor puede revisar de manera presencial las actividades.

El trabajo es en equipo de dos personas.

Ejercicio 6. Cálculo de velocidad de un motor
Ingeniería en Mecatrónica (O2023)

Objetivo:
A partir de un sensor óptico tipo herradura medir la velocidad de un motor.

Enunciado:
A partir del código realizado en clase. Realizar las siguientes actividades:

- Crear un proyecto
- Ajustar el código generado en clase para las siguientes funcionalidades
    - Hacer las interconexiones necesarias para conectar el sensor al microcontrolador
    - Haz los ajustes necesarios para que el sensor detecte el movimiento del motor
    - Ajustar el código visto en clase para medir la velocidad del motor usando input capture.
    - La velocidad debe de enviarse a la computadora vía UART, por interrupciones.
    - La velocidad se debe de visualizar en Matlab o Python y debe de haber una manera de corroborar la velocidad.
- Verificar la funcionalidad.

Materiales:

- Tarjeta TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Cable USB
- Code Composer Studio – IDE (CCS)
- Puente H para manejar un motor de corriente directa de manera bidireccional.
- Un motor de corriente directa
- Un sensor óptico tipo herradura
- Adaptación al motor para poder medir la velocidad

Entregable:
Se debe de subir a CANVAS el código a probar, agregarlo en un repositorio remoto y un video donde se realicen las actividades descritas en el enunciado. El profesor puede revisar de manera presencial las actividades.

El trabajo es en equipo de dos personas.

Ejercicio 10. Interfaz SPI
Ingeniería en Mecatrónica (O2023)

Objetivo:
Puesta en funcionamiento de un dispositivo con interfaz SPI

Enunciado:
A partir del código realizado en clase. Realizar las siguientes actividades:

- Crear un proyecto
- Ajustar el código generado en clase para las siguientes funcionalidades
    - Hacer las interconexiones necesarias para conectar el dispositivo SPI al microcontrolador
    - Ajustar el código visto en clase para realizar la comunicación vía SPI
    - La interfaz puede realizarse vía interrupciones o polling.
- Verificar la funcionalidad.

Materiales:

- Tarjeta TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Cable USB
- Code Composer Studio – IDE (CCS)
- Dispositivo con protocolo de comunicación SPI: se propone un potenciómetro digital.

Entregable:
Se debe de subir a CANVAS el código a probar, agregarlo en un repositorio remoto y un video donde se realicen las actividades descritas en el enunciado. El profesor puede revisar de manera presencial las actividades.

El trabajo es en equipo de dos personas.

Ejercicio 7. Interfaz IIC
(P2024)

Objetivo:
Puesta en funcionamiento de un dispositivo con interfaz IIC (MPU6050)

Enunciado:
A partir del código realizado en clase. Realizar las siguientes actividades:

- Crear un proyecto
- Ajustar el código generado en clase para las siguientes funcionalidades
    - Hacer las interconexiones necesarias para conectar el dispositivo IIC al microcontrolador
    - Ajustar el código visto en clase para realizar la comunicación vía IIC y poder medir la inclinación del sensor
- Verificar la funcionalidad.

Materiales:

- Tarjeta TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Cable USB
- Code Composer Studio – IDE (CCS)
- Dispositivo con protocolo de comunicación IIC (MPU6050)
- Módulo de comunicación BlueTooth HC06 o HC05

Entregable:
Se debe de subir a CANVAS el código a probar, agregarlo en un repositorio remoto y un video donde se realicen las actividades descritas en el enunciado. El profesor puede revisar de manera presencial las actividades.

El trabajo es en equipo de dos personas.

C. ENUNCIADOS DE LAS PRÁCTICAS

Práctica 1.
Ingeniería en Mecatrónica (O2023)

Objetivo:
Puesta en operación del control de velocidad y dirección de un carro de juguete.

Enunciado:

Desarrolla un sistema basado en el microcontrolador MSP430G2553 que cumpla con las siguientes características:

- A partir del movimiento de un volante, sensado con un potenciómetro, controlar un motor a pasos para definir la posición de las llantas delanteras del carro.
- A partir del movimiento de un pedal, mecánicamente similar a un acelerador de auto, y sensado por un potenciómetro, controlar la velocidad del motor de tracción (stepper).
- La prueba se realizará tendiendo el carro con las llantas al aire.

Materiales:

- Tarjeta TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Cable USB
- Code Composer Studio – IDE (CCS)
- Dos potenciómetros
- Dos motores a pasos

Entregable:
Se debe de subir a CANVAS el código fuente, agregado en un repositorio remoto y un video donde se realicen las actividades descritas en el enunciado.

Además de un reporte de práctica que incluya: el proceso de diseño y elaboración, el diagrama de interconexiones y el código documentado.

El profesor puede revisar de manera presencial las actividades.

El trabajo es en equipo de dos personas.

Práctica 2.
Ingeniería en Mecatrónica (O2023)

Objetivo:
Creación de un tablero de instrumentos para un auto de juguete.

Enunciado:

Desarrolla un sistema basado en el microcontrolador MSP430G2553 que cumpla con las siguientes características:

- Se deben de sensar la velocidad del auto, el nivel de batería, y la posición GPS de un auto de juguete.
- La comunicación se realizará usando una UART y un módulo de comunicación inalámbrica
- Los datos se desplegarán en una aplicación desarrollada en ambiente Python en una PC.
- Se deberá verificar que la información del GPS es correcta.

Materiales:

- Tarjeta TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Cable USB
- Code Composer Studio – IDE (CCS)
- Dos potenciómetros
- Dos motores a pasos
- Un sensor de velocidad
- Un sensor para medir el nivel de batería
- Un GPS con interfaz serial.
- Un módulo de comunicación inalámbrica
- Programa en lenguaje Python para PC

Entregable:
Se debe de subir a CANVAS el código fuente y un video donde se realicen las actividades descritas en el enunciado. Además de un reporte de práctica que incluya: el proceso de diseño y construcción de la práctica, diagrama de interconexiones, así como el código comentado y agregado en un repositorio remoto. Se deben incluir captura y análisis de las tramas de sensores con ayuda del analizar lógico

El profesor puede revisar de manera presencial las actividades.

El trabajo es en equipo de dos personas.

Práctica 3.
Ingeniería en Mecatrónica (O2023)

Objetivo:

Integración al auto de juguete de un sensor angular, aceleración y comunicación inalámbrica a distancia media.

Enunciado:

Desarrolla un sistema basado en el microcontrolador MSP430G2553 que cumpla con las siguientes características:

- Se deben de sensar la aceleración del auto y el ángulo contra la horizontal. El sensor tendrá como interfaz el protocolo IIC.
- Los datos del auto se comunicarán con un módulo inalámbrico con protocolo SPI.
- Los datos se recibirán por un MSP430 vía SPI y se enviarán por protocolo UART a una computadora para su visualización.
- Los datos se desplegarán en una aplicación desarrollada en ambiente Python en una PC.
- Se deberá verificar que la información es correcta.

Materiales:

- Dos tarjetas TI LaunchPad kit with MSP430 MCU (MSP-EXP430G2ET)
- Dos cables USB
- Code Composer Studio – IDE (CCS)
- Dos potenciómetros
- Dos motores a pasos
- Un sensor angular y de aceleración (protocolo IIC)
- Dos módulos de comunicación inalámbrica de distancia media (protocolo SPI)
- Programa en lenguaje Python para PC

Entregable:

Se debe de subir a CANVAS el código fuente y un video donde se realicen las actividades descritas en el enunciado. Además de un reporte de práctica que incluya: el proceso de diseño y construcción de la práctica, diagrama de interconexiones, así como el código comentado y agregado en un repositorio remoto. Se deben incluir captura y análisis de las tramas de sensores con ayuda del analizar lógico

El profesor puede revisar de manera presencial las actividades.

El trabajo es en equipo de dos personas.

D. ENUNCIADO DE PROYECTO FINAL

Proyecto final.
Ingeniería en Mecatrónica (O2023)

Objetivo:
Integración de conocimientos y habilidades aprendidas en el curso para desarrollar un sistema que permita conducir un vehículo de juguete a distancia

Enunciado:

Desarrolla un sistema basado en el microcontrolador MSP430G2553 que cumpla con las siguientes características:

- A partir del movimiento de un volante, sensado con un potenciómetro, controlar a distancia la dirección de las llantas delanteras del carro.
- A partir del movimiento de un pedal, mecánicamente similar a un acelerador de auto, y sensado por un potenciómetro, controlar a distancia la velocidad del motor de tracción del carro.
- Se deben de sensar, a distancia, la velocidad, el nivel de batería, y la posición GPS de un auto de juguete. Los datos se desplegarán en una aplicación desarrollada en ambiente Python en una PC. Se deberá verificar que la información del GPS es correcta.
- Los datos entre ambos sistemas se comunicarán usando un módulo nRF2401 con protocolo SPI.
- Los datos se recibirán por un MSP430 vía SPI y se enviarán por protocolo UART a una computadora para su visualización.
- Se deberá verificar que la información es correcta.
- Se debe de instalar una cámara al vehículo y se debe de visualizar la imagen de la cámara de manera remota en la PC.

Entregables:
Se debe de subir a CANVAS un video donde se realicen las actividades descritas en el enunciado. Además de un reporte de proyecto que incluya: el proceso de diseño y construcción del proyecto, diagrama de interconexiones, así como el código comentado.

El profesor revisará de manera presencial la demostración de la funcionalidad del sistema.

El trabajo es en equipo de cuatro personas máximo.

E. Programas de referencia en lenguaje C

```
//E.1 Código para prender y apagar un led de la tarjeta de desarrollo, usando una
// subrutina de delay basada en un contador simple

#include <msp430.h>

void delay (void)
{
volatile unsigned short i;        // volatile to prevent optimization

for (i=0;i<=50000;i++);
}

void main(void)
{
   WDTCTL = WDTPW | WDTHOLD;       // stop watchdog timer

   P1SEL=0;
   P1SEL2=0;                        // configura pin como GPIO, valor de Reset
   P1DIR = 0x01;                 // configure P1.0 as output

   while(1)
   {
      P1OUT=1;                           // prender led
          delay();
          P1OUT=0;                       // apagar led
          delay();
   }
}
```

```
//E2. Código para mover un motor a pasos. El tiempo entre los pasos se define con un delay generado
// por un timer en modo básico.

#include <msp430.h>
void delay_ms (unsigned int tiempo_ms)
{
unsigned short tiempo;

      tiempo=tiempo_ms;
     do{
         TACCR0 = TAR + 1000;
         do {} while ((TACCTL0 & BIT0) == 0);
         TACCTL0&=~(BIT0);
      }while (--tiempo);
}

void main(void)
{
   WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer
   TACTL = TASSEL_2 + MC_2; // SMCLK, contmode
   P2DIR|=BIT3+BIT2+BIT1+BIT0;
  while (1)
  {
       P2OUT=12; // 1100
       delay_ms(2);
       P2OUT=6; // 0110
      delay_ms(2);
       P2OUT=3; // 0011
       delay_ms(2);
       P2OUT=9; // 1001
      delay_ms(2);
  }
}
```

```
//E3. Código para manejar un control a pasos usando un timer para generar el tiempo entre los
//elementos de la secuencia. Incluyendo un pin de entrada para definir el sentido de giro

#include <msp430.h>
unsigned char stepper_sec[]={12,6,3,9};

void delay_ms (unsigned int tiempo_ms)
{
   unsigned short tiempo;

   tiempo=tiempo_ms;
   do{
      TACCR0 = TAR + 1000;
      do {} while ((TACCTL0 & BIT0) == 0);
      TACCTL0&=~(BIT0);
   }while (--tiempo);
}

void main(void)
{
  unsigned char paso=0;
   WDTCTL = WDTPW | WDTHOLD;          // stop watchdog timer
   TACTL = TASSEL_2 + MC_2;           // SMCLK, contmode

   P2DIR|=BIT3+BIT2+BIT1+BIT0;

   while (1)
   {
      if ((P1IN & BIT3)!=0) P2OUT=stepper_sec[paso++%4];
      else                  P2OUT=stepper_sec[paso--%4];
      delay_ms(2);
   }
}
```

//E4. Código para manejar un motor a pasos usando interrupciones del *TimerA*

```
#include <msp430.h>
unsigned char stepper_sec[]={12,6,3,9};
unsigned int tiempo_stepper_reload;
unsigned char paso=0;

// Timer A0 interrupt service routine

#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
  TACCR0 += tiempo_stepper_reload;          // Add Offset to CCR0
  if ((P1IN & BIT3)!=0) P2OUT=stepper_sec[paso++%4];
  else P2OUT=stepper_sec[paso--%4];
}

void Stepper_time_us (unsigned int tiempo_stepper_us)
{
   TACCTL0 |= CCIE;                         // CCR0 interrupt enable
   TACCR0 = TAR + tiempo_stepper_us;
   tiempo_stepper_reload=tiempo_stepper_us;
   TACTL = TASSEL_2 + MC_2;                 // SMCLK, contmode

   __bis_SR_register(GIE);                  // Enable interrupt
}

void main(void)
{
   WDTCTL = WDTPW | WDTHOLD;                // stop watchdog timer

   P2DIR|=BIT3+BIT2+BIT1+BIT0;
   Stepper_time_us (2000);                  // 16 bits range

   while (1);
}
```

```
//E5. Código para leer un pin de entrada del ADC donde con un potenciómetro se varía el voltaje
//Se activa un led si el voltaje está por debajo de un umbral.

#include <msp430.h>

#define V_umbral_mV 1650ul                          //1.65 V (1650 mV)
#define ADC_umbral (V_umbral_mV*1024)/3300 //Vref: 3.3 V (3300 mv), ADC de 10 bits:
                                                    //2^10=1024

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
   if (ADC10MEM < ADC_umbral)
        P1OUT &= ~0x01;                  // Clear P1.0 LED off
   else
     P1OUT |= 0x01;                   // Set P1.0 LED on

   ADC10CTL0 |= ADC10SC;               // Conversion start
}

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;              // Stop WDT

  ADC10CTL0 = ADC10ON + ADC10IE;   //  ADC10ON, interrupt enabled. VR+: Vcc.
                                   //VR-: Vss
  ADC10CTL1 = INCH_1;              // input A1
  ADC10AE0 |= 0x02;                // PA.1 ADC option select

  P1DIR |= 0x01;                    // Set P1.0 to output direction

  ADC10CTL0 |= ENC + ADC10SC;       // Sampling and conversion start
  __bis_SR_register(GIE);           // GIE=1

  while (1);

}
```

//E6. Código para leer varias muestras de la entrada A2 del ADC

```
#include <msp430.h>

unsigned int ADC_samples[50];

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL1 = CONSEQ_2 + INCH_2;            // Repeat single channel, A2
  ADC10CTL0 = MSC + ADC10ON + ADC10IE; // ADC10ON, interrupt enable
  ADC10DTC1 = 32;                           // 32 conversions
  ADC10AE0 |= 0x04;                         // P1.2 ADC option select

  for (;;)
  {
    ADC10SA = (unsigned int)&ADC_samples;       // Data buffer start
    ADC10CTL0 |= ENC + ADC10SC;                 // Sampling and conversion start
    __bis_SR_register(CPUOFF + GIE);            // LPM0, ADC10_ISR will force exit
  }
}
```

//E7. Código para leer de tres entradas analógicas en secuencia

```
#include <msp430.h>

unsigned int ADC_samples[50];
int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL1 = INCH_4 + CONSEQ_1;            // A4/A3/A2, single sequence
  ADC10CTL0 = MSC + ADC10ON + ADC10IE;
  ADC10DTC1 = 0x03;                         // 3 conversions
  ADC10AE0 |= 0x1C;                         // P1.3,2,1 ADC10 option select
  P1DIR |= 0x01;                            // Set P1.0 output

  for (;;)
  {
    ADC10CTL0 &= ~ENC;
    ADC10SA = (unsigned int)&ADC_samples;   // Data buffer start
    P1OUT |= 0x01;                          // P1.0 = 1
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
    __bis_SR_register(CPUOFF + GIE);        // LPM0, ADC10_ISR will force exit
    P1OUT &= ~0x01;                         // P1.0 = 0
  }
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}
```

//E8. Código para mandar un dato por UART: 9600 bps, 8 bits, 1 stop bit, sin paridad

```
#include <msp430.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

  P1SEL = BIT1 + BIT2 ;                     // P1.1 = RXD, P1.2=TXD
  P1SEL2 = BIT1 + BIT2 ;                    // P1.1 = RXD, P1.2=TXD

  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 = 104;                  // UCA0BR1:UCA0BR0= SMCLK/baud rate= 1MHz/9600
  UCA0BR1 = 0;

  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**

  UCA0TXBUF = 'A';
  while (1);
}
```

//E9. Código para mandar de A a Z por UART, necesario verificar la bandera de tranmisión

```
#include <msp430.h>

void main(void)
{
unsigned char i;

  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

  P1SEL = BIT1 + BIT2 ;                     // P1.1 = RXD, P1.2=TXD
  P1SEL2 = BIT1 + BIT2 ;                    // P1.1 = RXD, P1.2=TXD

  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 = 104;                      // UCA0BR1:UCA0BR0= SMCLK/baud rate= 1MHz/9600
  UCA0BR1 = 0;                       /

  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**

  for(i='A'; i<='Z';i++)
  {
     while ((IFG2 & UCA0TXIFG)==0);         // USCI_A0 TX buffer ready?
     UCA0TXBUF = i;                    //
  }

  while (1);
}
```

//E10. Código para mandar una cadena predefinida en un arreglo

```
#include <msp430.h>

unsigned char mensaje1[]={"Hola mundo"};

void main(void)
{
unsigned char i=0;

  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

  P1SEL = BIT1 + BIT2 ;                     // P1.1 = RXD, P1.2=TXD
  P1SEL2 = BIT1 + BIT2 ;                    // P1.1 = RXD, P1.2=TXD

  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 = 104;                  // UCA0BR1:UCA0BR0= SMCLK/baud rate= 1MHz/9600
  UCA0BR1 = 0;

  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**


  do{
     while ((IFG2 & UCA0TXIFG)==0);         // USCI_A0 TX buffer ready?
     UCA0TXBUF = mensaje1[i++];             //
  }while (mensaje1[i]!=0);

  while (1);
}
```

//E11.Código para mandar por UART, integrando una función para convertir de entero a ASCIIs

```
#include <msp430.h>

unsigned char *Ptr_string;
unsigned char UART_TX_pending=0;
unsigned int ADC_mV;
unsigned char volt_string[]={"Valor=#.### V\n\r"};

#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void)
{
 if (*Ptr_string!=0) UCA0TXBUF = *Ptr_string++;
 else
 {
    IE2 &= ~UCA0TXIE;                       // Enable USCI_A0 RX interrupt
    UART_TX_pending=0;
 }
}

void UART_TX_send_init (unsigned char *string)
{
   Ptr_string=string;
   UART_TX_pending=1;
   IE2 |= UCA0TXIE;                       // Enable USCI_A0 RX interrupt
}

void main(void)
{
 WDTCTL = WDTPW + WDTHOLD;               // Stop WDT

 P1SEL = BIT1 + BIT2 ;                   // P1.1 = RXD, P1.2=TXD
 P1SEL2 = BIT1 + BIT2 ;                   // P1.1 = RXD, P1.2=TXD

 UCA0CTL1 |= UCSSEL_2;                   // SMCLK
 UCA0BR0 = 104;                        // UCA0BR1:UCA0BR0= SMCLK/baud rate= 1MHz/9600
 UCA0BR1 = 0;                          //
 dutycycle1,dutycycle2 = 100;
 UCA0CTL1 &= ~UCSWRST;                   // **Initialize USCI state machine**

 __bis_SR_register(GIE);                 // Enter LPM0, interrupts enabled

 ADC_mV=1650;
 volt_string[10]=(ADC_mV%10)+0x30;
 ADC_mV=ADC_mV/10;
```

```
  volt_string[9]=(ADC_mV%10)+0x30;
  ADC_mV/=10;
  volt_string[8]=(ADC_mV%10)+0x30;
  volt_string[6]=(ADC_mV/10)+0x30;

  do{}while (UART_TX_pending==1);
  UART_TX_send_init((unsigned char *) volt_string);

  while (1);}//Código para recibir un dato por UART y reenviarlo sumándolo un valor 1

#include <msp430.h>

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

  P1SEL = BIT1 + BIT2 ;                     // P1.1 = RXD, P1.2=TXD
  P1SEL2 = BIT1 + BIT2 ;                    // P1.1 = RXD, P1.2=TXD
  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 = 104;                            // 1MHz 9600
  UCA0BR1 = 0;

  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**

  while(1)
  {
     while ((IFG2 & UCA0RXIFG)==0);         // USCI_A0 RX buffer ready?
     IFG2&= ~UCA0RXIFG;
     UCA0TXBUF = UCA0RXBUF+1;                // TX -> RXed character +1
  }
}
```

```
//E12. Código que recibe un conjunto de ASCIIs asociados a los números y genera la conversión
// atoi. Usando polling

#include <msp430.h>

void main(void)
{
unsigned char temp;
volatile unsigned int valor;

WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

  P1SEL = BIT1 + BIT2 ;                   // P1.1 = RXD, P1.2=TXD
  P1SEL2 = BIT1 + BIT2 ;                  // P1.1 = RXD, P1.2=TXD
  UCA0CTL1 |= UCSSEL_2;                   // SMCLK
  UCA0BR0 = 104;                          // 1MHz 9600
  UCA0BR1 = 0;

  UCA0CTL1 &= ~UCSWRST;                   // **Initialize USCI state machine**

  while(1)
  {
     valor=0;
     do{
        while ((IFG2 & UCA0RXIFG)==0);          // USCI_A0 RX buffer ready?
        //IFG2&= ~UCA0RXIFG;

        temp = UCA0RXBUF;
        if ((temp>='0') && (temp<='9')) valor=valor*10+temp-0x30;
     }while (temp!=13);
  }
}
```

```
//E13. Código que recibe un conjunto de ASCIIs por UART y los convierte a número (atoi),
usando
//interrupciones

#include <msp430.h>

volatile unsigned int valor=0;
volatile unsigned char numero_recibido=0;

#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void)
{
unsigned char temp;

   temp = UCA0RXBUF;
   if ((temp>='0') && (temp<='9')) valor=valor*10+temp-0x30;

   if (temp==0x0D)
   {
      numero_recibido=1;
      valor=0;                                  //debería de manejarse fuera de la intr
   }
}


void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

  P1SEL = BIT1 + BIT2 ;                     // P1.1 = RXD, P1.2=TXD
  P1SEL2 = BIT1 + BIT2 ;                    // P1.1 = RXD, P1.2=TXD
  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 = 104;                            // 1MHz 9600
  UCA0BR1 = 0;

  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**

  IE2 |= UCA0RXIE;                          // Enable USCI_A0 RX interrupt

  __bis_SR_register(GIE);                   // interrupts enabled

  while(1);
}
```

//E14. Código IIC

```
#include <msp430.h>

#define I2CSEL   P1SEL
#define I2CSEL2  P1SEL2
#define I2CSCL   BIT6
#define I2CSDA   BIT7

typedef unsigned char uint8_t;

uint8_t *PTxData;                     // Pointer to TX data
uint8_t *PRxData;                     // Pointer to RX data

uint8_t TxByteCtr;
uint8_t RxByteCtr;

const unsigned char TxData[] =              // Table of data to transmit
{
  0,0x25
};

void I2C_init(uint8_t slaveAddress){
    // Port Configuration
    I2CSEL |= I2CSDA + I2CSCL;                    // Assign I2C pins to USCI_B0
    I2CSEL2|= I2CSDA + I2CSCL;                    // Assign I2C pins to USCI_B0

//    isRx = 0;                                   // State variable - possibly
useless

    //USCI Configuration
    UCB0CTL1 |= UCSWRST;                          // Enable SW reset
    UCB0CTL0 = UCMST + UCMODE_3 + UCSYNC;         // I2C Master, synchronous mode
    UCB0CTL1 = UCSSEL_2 + UCSWRST;                // Use SMCLK, keep SW reset

    //Set USCI Clock Speed
    UCB0BR0 = 12;                                 // fSCL = SMCLK/12 = ~100kHz
    UCB0BR1 = 0;

    //Set Slave Address and Resume operation
    UCB0I2CSA = slaveAddress;                     // Slave Address passed as
parameter
    UCB0CTL1 &= ~UCSWRST;                         // Clear SW reset, resume
operation
}


void I2C_write(uint8_t ByteCtr, uint8_t *TxData) {
    __disable_interrupt();
//    isRx = 0;
    //Interrupt management
    IE2 &= ~UCB0RXIE;                             // Disable RX interrupt
//    while (UCB0CTL1 & UCTXSTP);                 // Ensure stop condition got sent
```

```
    IE2 |= UCB0TXIE;                                // Enable TX interrupt

    //Pointer to where data is stored to be sent
    PTxData = (uint8_t *) TxData;                   // TX array start address
    TxByteCtr = ByteCtr;                            // Load TX byte counter

    //Send start condition
    //    while (UCB0CTL1 & UCTXSTP);               // Ensure stop condition got sent
    UCB0CTL1 |= UCTR + UCTXSTT;                     // I2C TX, start condition

    __bis_SR_register(CPUOFF + GIE);                // Enter LPM0 w/ interrupts
    while (UCB0CTL1 & UCTXSTP);
}

void I2C_read(uint8_t ByteCtr, volatile uint8_t *RxData) {
    __disable_interrupt();
//    isRx = 1;

    //Interrupt management
    IE2 &= ~UCB0TXIE;                               // Disable TX interrupt
    UCB0CTL1 = UCSSEL_2 + UCSWRST;                  // Use SMCLK, keep SW reset
    UCB0CTL1 &= ~UCSWRST;                           // Clear SW reset, resume
operation
    IE2 |= UCB0RXIE;                                // Enable RX interrupt

    //Pointer to where data will be stored
    PRxData = (uint8_t *) RxData;                   // Start of RX buffer
    RxByteCtr = ByteCtr;                            // Load RX byte counter

    //while (UCB0CTL1 & UCTXSTP);                   // Ensure stop condition got sent

    //If only 1 byte will be read send stop signal as soon as it starts transmission
    if(RxByteCtr == 1){
        UCB0CTL1 |= UCTXSTT;                        // I2C start condition
        while (UCB0CTL1 & UCTXSTT);                 // Start condition sent?
        UCB0CTL1 |= UCTXSTP;                        // I2C stop condition
        __enable_interrupt();
    } else {
        UCB0CTL1 |= UCTXSTT;                        // I2C start condition
    }

    __bis_SR_register(CPUOFF + GIE);                // Enter LPM0 w/ interrupts
    while (UCB0CTL1 & UCTXSTP);                     // Ensure stop condition got sent
}

void main (void)
{
    I2C_init(0x50);
    PTxData = (unsigned char *)TxData;      // TX array start address
    //TXByteCtr = sizeof TxData;              // Load TX byte counter
    I2C_write (sizeof TxData, PTxData );

}
```

```
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector = USCIAB0TX_VECTOR
__interrupt void USCIAB0TX_ISR(void)
{
  if(IFG2 & UCB0RXIFG){                              // Receive In
  if (RxByteCtr == 1)
  {
     *PRxData = UCB0RXBUF;                           // Move final RX data to PRxData
     __bic_SR_register_on_exit(CPUOFF);              // Exit LPM0
  }
  else
  {
      *PRxData++ = UCB0RXBUF;                        // Move RX data to address
PRxData
      if (RxByteCtr == 2)                            // Check whether byte is second
to last to be read to send stop condition
      UCB0CTL1 |= UCTXSTP;
      __no_operation();
  }
  RxByteCtr--;                                       // Decrement RX byte counter
  }

  else{                                              // Master Transmit
      if (TxByteCtr)                                 // Check TX byte counter
  {
    UCB0TXBUF = *PTxData;                            // Load TX buffer
    PTxData++;
    TxByteCtr--;                                     // Decrement TX byte counter
  }
  Else999999
  {
    UCB0CTL1 |= UCTXSTP;                             // I2C stop condition
    IFG2 &= ~UCB0TXIFG;                              // Clear USCI_B0 TX int flag
    __bic_SR_register_on_exit(CPUOFF);               // Exit LPM0
  }
 }
}
```

//E15. Código para leer un potenciómetro digital usando SPI

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer

  P1OUT |= BIT3;                            // CS init value (1)
  P1DIR |= BIT3;                            // CS Output

  P1SEL |= BIT1 + BIT2 + BIT4;
  P1SEL2 |= BIT1 + BIT2 + BIT4;
  UCA0CTL0 |= UCCKPH + UCCKPL+ UCMSB + UCMST + UCSYNC;  // 3-pin, 8-bit SPI
                                                        //master
  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 = 1;                              // /1
  UCA0BR1 = 0;
  UCA0MCTL = 0;                             // No modulation
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**

  P1OUT &= ~(BIT3);                         // /CS=0
  __delay_cycles(5);                        // Wait for slave to initialize

  UCA0TXBUF = 0;                            // Transmit first character (High byte)
  while (!(IFG2 & UCA0TXIFG));              // USCI_A0 TX buffer ready?

  UCA0TXBUF = 0x10;                         // Transmit second character (Low byte)
  while (!(IFG2 & UCA0TXIFG));              // USCI_A0 TX buffer ready?

  P1OUT |= BIT3;                            // /CS=1

  __bis_SR_register(LPM0_bits);             // CPU off
}
```

```
//E16. Código que genera un PWM usando la funcionalidad en hardware y genera un
frecuencímetro usando la funcionalidad de input capture

#include <msp430.h>

#define periodo_1ms 1000
unsigned char duty_cycle;

unsigned char Count, First_Time;
unsigned int REdge1, REdge2, FEdge;

unsigned int Period, ON_Period;
unsigned char DutyCycle;

// TA0_A1 Interrupt vector
#pragma vector = TIMER0_A1_VECTOR
__interrupt void TIMER0_A1_ISR (void)

{
  switch(TA0IV)
  {
     case  TA0IV_NONE: break;                    // Vector  0:  No interrupt
     case  TA0IV_TACCR1:                         // Vector  2:  TACCR1 CCIFG
       if (TA0CCTL1 & CCI)                       // Capture Input Pin Status
       {
          // Rising Edge was captured
          if (!Count)
          {
             REdge1 = TA0CCR1;
             Count++;
          }
          else
          {
             REdge2 = TA0CCR1;
             Count=0x0;
             __bic_SR_register_on_exit(LPM0_bits + GIE);  // Exit LPM0 on return to main
          }

          if (First_Time)
             First_Time = 0x0;
       }
       else
       {
          // Falling Edge was captured
          if(!First_Time)
```

```
          {
            FEdge = TA0CCR1;
          }
        }
        break;
      case TA0IV_TACCR2: break;                 // Vector  4:  TACCR2 CCIFG
      case TA0IV_6: break;                      // Vector  6:  Reserved CCIFG
      case TA0IV_8: break;                      // Vector  8:  Reserved CCIFG
      case TA0IV_TAIFG: break;                  // Vector 10:  TAIFG
      default:  break;
  }
}

void PWM_start (unsigned char duty_cycle)
{
   // Configure Port Pins Data Sheet (page 43 - )
   P2DIR |= BIT1;                               // P2.1/TA1.1 Output
   P2SEL |= BIT1;                               // TA1.1 Option select

   // Configure TA1.1 to output PWM signal
   // Period = 1000/1 MHz = 1 ms ~ 1000Hz Freq
   TA1CCR0 = periodo_1ms-1;                     // Period Register
   TA1CCR1 = duty_cycle*10;                     // TA1.1 dutycycle
   TA1CCTL1 |= OUTMOD_7;                        // TA1CCR1, Reset/Set
   TA1CTL = TASSEL_2 + MC_1 + TACLR;            // SMCLK (1 MHz), upmode, clear TAR
}

void timed_input_measurement_init (void)
{
   // Configure Port Pins Data Sheet (page 43 - )
   P1DIR &= ~BIT2;                              // P1.2/TA0.1 Input Capture
   P1SEL |= BIT2;                               // TA0.1 option select

   // Configure the TA0CCR1 to do input capture
   TA0CCTL1 = CM_3 + SCS + CAP +  CCIE;
                                // Both Rising and Falling Edge; Synchronous;
                                //  TA0CCR1 Capture mode; interrupt enable
   TA0CTL |= TASSEL_2 + MC_2 + TACLR;       // SMCLK, Cont Mode; start timer
}

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer

  PWM_start (25);
```

```
  timed_input_measurement_init();


  // Variable Initialization
  Count = 0x0;
  First_Time = 0x01;

  while(1)
  {
     __bis_SR_register(LPM0_bits + GIE);          // Enter LPM0
     __no_operation();                            // For debugger
     Period = REdge2 - REdge1;                    // Calculate Period
     ON_Period = FEdge-REdge1;                    // On period
     DutyCycle = ((unsigned long)ON_Period*100/Period);
  }
}
```

Índice