UCLV
Universidad Central
"Marta Abreu" de Las Villas

FIE
Facultad de
Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Título: Prácticas de laboratorio con microcontrolador ATmega88PA

Autor: José Raúl Conde Pérez

Tutores: M.Sc. Robby Gustabello Cogle

Ing. Guillermo Ambar Pérez

Santa Clara noviembre, 2020

UCLV
Universidad Central
"Marta Abreu" de Las Villas

FIE
Facultad de
Ingeniería Eléctrica

Department of Automation and Computer Systems

TRABAJO DE DIPLOMA

Title: ATmega88PA microcontroller labs

Author: José Raúl Conde Pérez

Thesis Director: M.Sc. Robby Gustabello Cogle

Ing. Guillermo Ambar Pérez

Santa Clara, noviembre, 2020

PENSAMIENTO

Que algo no haya salido como hayas querido no significa que sea inútil.

Thomas Edison

DEDICATORIA

Este trabajo lo dedico a mis padres quienes se han sacrificado para darme todo lo que han tenido en todo el transcurso de mi vida, a mi hermano, a mi novia, a mi tío Ernesto, a mi abuela Nilda, a Emma y a Ricardo, a Lili, Idalmis y a Osmel, a mi padrino y JuanPa, a todos mis amigos y a Adrián de La Paz Echemendía.

AGRADECIMIENTOS

- A Dios.
- A mis tutores Robby Gustabello Cogle y a Guillermo Ambar Pérez.
- A la profesora María del Carmen no solo por sus magistrales clases sino por sus consejos para la vida estudiantil y sobre todo los consejos que me dió para la vida real.
- A mis eléctricos.
- A mis amigos en especial a Dioni, Javier Castellón, Javier Lozano, Jorge Eduardo Cossio, Lázaro Fortún (Lachy), Frank Emilio, Danay Rivero Rivera, Ailan Alcántara, Ramón, Alienqui Aguilar Salvador, Yadier Sánchez Martín, Alejandro Barón Fiss, Juan Carlos García García y Odaimis Márquez Cáceres por sus constantes preocupaciones por sus consejos y por los momentos compartidos tanto los de pena como los de gloria.
- A mi padrino Tomás Faife por devolverme la fe y el pensamiento de creer en mi cuando pensé que todo estaba perdido.
- A Osmel e Idalmis por demostrarme que soy el hijo varón que no tuvieron incluso hasta los últimos momentos antes de la graduación ustedes mostraron alta preocupación, sacrificio y entrega hacia mi persona y dejándome bien claro que el puro es el puro y eso no se me va a olvidar jamás.
- A Lily por clavarse en mi corazón con ese carisma que tiene y que sepas que siempre vas a ser mi hermanita del alma.
- A Emma Orellana y a Ricardo Morejón por acogerme como un verdadero hijo incluso sin tan siquiera conocerme, honestamente gracias por sus consejos y apoyo en los momentos difíciles, eso tampoco será olvidado.
- A mi tío Ernesto por ser tan especial, carismático, preocupado, entregado, y por pensar en mí todos los días de este mundo. Por eso usted es para mí un padre y lo voy a querer siempre como tal, y no te preocupes, el que no te quiera no sabe cuánto se está perdiendo.

- A mi abuela Nilda por mimarme y dármelo todo, por convertirme en su nieto más consentido y por todos estos años de preocupación momentos difíciles tus consejos siempre fueron escuchados. Por todo quiero decirte gracias mi “Tacañona”
- A mi hermano por nunca reprocharme nada, por cambiarme mi vida desde la primera vez que te vi tanto así, que desde ese momento pasaste a ser la persona que más quiero en el mundo, nunca pensé que el amor de hermanos fuese tan grande, por eso pase lo que pase vas a ser siempre mi persona más consentida.
- A mi novia Claudia Morejón Orellana por ser mi gran bastón en los momentos más difíciles, incluso en los momentos donde no había nadie más, ahí estabas para decirme con esa luz que tienes, que aleja todo mal, que todo iba a salir, por darme confianza en mi mismo y sobre todas las cosas por darme ese inmenso amor que cada día hace que me enamore más y más de ti, desearnos que duremos muchísimos años tanto en esta vida como en la otra en una vida de infinito amor y felicidad.
- A mis padres Elaine Pérez Yero y José Raúl Conde Collado, a ustedes no me alcanzan las palabras y a veces no las encuentro para expresar lo que significan para mí, por eso primeramente quisiera decirle que ustedes son dioses, en cuanto lo que a mi refiere, que sin que me quede nada por dentro ustedes son los mejores padres que hay en el mundo, nunca he visto tanto sacrificio como el que ustedes hacen por sus hijos, los amo mucho mucho, gracias por la educación que me han dado, por sus consejos, por sus alones de orejas, por su amor, por su entrega y por llenarme de conceptos que en momentos de mi vida yo he llegado a envidiarlos. Son innumerables las razones para agradecerle por eso le pido a la vida que me deje devolverle al menos una mínima parte de lo que me han dado, digo mínima porque igualarlo es muy difícil y superarlos prácticamente imposible. Simplemente muchas gracias por traerme al mudo y formarme en el hombre que soy hoy.

RESUMEN

El acelerado desarrollo tecnológico actual ha exigido la modernización en prácticamente todas las esferas de la sociedad y la carrera de Ingeniería en Automática no escapa de este fenómeno. Es por ello que la misma se encuentra llevando a cabo un serio período de actualización en aras de mejorar la formación de los estudiantes. Basado en ello, el objetivo central del trabajo radica en el diseño de nuevas prácticas de laboratorio relacionadas con los microcontroladores.

Se proponen cuatro prácticas que responden a aspectos básicos del trabajo con los microcontroladores, específicamente con aquellos de la familia ATmega. Se destaca además el empleo por vez primera de Atmel Studio 7 en la solución de las prácticas cuyo montaje real es fácil de implementar. Con ello se profundiza el ejercicio con esta poderosa herramienta computacional, lo que resulta altamente novedoso. El resultado final del trabajo fue la ejecución y comprobación de cada una de estas prácticas.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Los microcontroladores son sumamentes importantes hoy en día, ya que constituyen una tecnología popular de nuestra era. Mientras que algunas décadas atrás parecía extremadamente difícil enseñar a personas que no eran ingenieros eléctricos como programar un sistema computacional de un solo chip, en la actualidad, diseñadores sin experiencia comparten prototipos basados en microcontroladores en internet. El surgimiento de la cultura actual ha introducido la permanencia de esta tecnología e hizo que el desarrollo de los sistemas informáticos integrados sea más fácil que nunca (Bolanakis, 2019).

La tecnología de microcontroladores pertenece a la categoría de sistemas informáticos integrados que siguen las reglas del método de programación convencional (secuencial) y no ejecutan un sistema operativo (SO). El último atributo diferencia a los microcontroladores de la tecnología aliada de microprocesadores. El efecto de "hacerlo uno mismo" en la tecnología de microprocesadores ha dado forma a las computadoras de una sola placa de hoy (como la placa Raspberry Pi), donde el sistema operativo generalmente se carga en una tarjeta digital segura (SD) integrada. Si bien la programación y el desarrollo de la aplicación de una computadora de una sola placa se pueden realizar de manera similar a la de las computadoras personales (PC), el procedimiento correspondiente para un sistema basado en microcontroladores conlleva desafíos particulares. Los desafíos están de acuerdo con el control de hardware de bajo nivel que el sistema operativo oculta regularmente en una PC o computadora de una sola placa. Debido a esta diferenciación en el desarrollo de aplicaciones con microcontroladores, junto con la posibilidad de implementar abundantes aplicaciones fascinantes, la formación de microcontroladores ha ganado terreno últimamente y continúa atrayendo cada vez más atención (Bolanakis, 2019).

Existen varios fabricantes de microcontroladores, uno de los que se destaca en la actualidad es Atmel[1]. Su línea de productos contiene microcontroladores, dispositivos de radiofrecuencia, memorias de solo lectura programable y borrable eléctricamente (EEPROM por sus siglas en inglés), flash y muchas otras. Atmel sirve a los mercados de la electrónica de consumo, comunicaciones, computadores, redes, electrónica industrial, equipos médicos, automotriz, aeroespacial y militar. Es una industria líder en sistemas seguros (Molero Prieto, 2016).

Los microcontroladores megaAVR provienen de la tecnología Atmel y han sido empleados en varios campos que requieren tecnología. Uno de ellos es la industria automovilística, la cual es muy amplia y se extiende a casi todo el planeta (Atmel, 2008). Entre sus principales productores mundiales se encuentra la industria automovilística alemana la cual cuenta con varias marcas de renombre, por ello la Universidad de Heilbronn ha incentivado el estudio de dichos microcontroladores. Una de las formas que han implementado es la introducción del módulo µECU (Micro ECU) en su plan de estudio. Y, por el intercambio de conocimientos y superación de los profesores y estudiantes entre Cuba y Alemania, específicamente entre la universidad de Heilbronn y la Universidad Central “Marta Abreu” de Las Villas (UCLV), la carrera de Ingeniería Automática de la Facultad de Ingeniería Eléctrica se ha dotado de varios módulos para mejorar el desarrollo de habilidades, facilitar el aprendizaje y aumentar el conocimiento por parte de los estudiantes con una tecnología primermundista.

El módulo µECU se basa en el microcontrolador ATmega88PA perteneciente a la familia megaAVR, el módulo incluye además sensores, botones, una pantalla de cristal líquido (LCD por sus siglas en inglés), etc. Teniendo en cuenta la situación existente, se plantea como problema científico: el deficiente manejo de los microcontroladores ATmega por parte de los estudiantes de Automática de la Universidad Central "Marta Abreu" de Las Villas.

Para dar solución se traza el objetivo general: Desarrollar prácticas de laboratorio con el módulo µEcu, para facilitar la adquisición de habilidades en el manejo de microcontroladores

[1] Acrónimo en inglés de “tecnología avanzada para la memoria y la lógica”.

ATmega por parte de los estudiantes de Automática de la Universidad Central "Marta Abreu" de Las Villas.

Objetivos específicos:

- Analizar la bibliografía disponible relativa a la tecnología Atmel.
- Revisar los fundamentos teóricos que sustentan la realización de prácticas de laboratorio.
- Describir las funcionalidades de Atmel Studio 7 como software de programación de los microcontroladores Atmel.
- Proponer las prácticas de laboratorio a desarrollar.
- Implementar la solución de cada práctica y probar su correcto funcionamiento.

Es necesario que los estudiantes, además de adquirir los fundamentos teóricos que llevan al conocimiento de esta nueva tecnología, finalicen entendiendo con mejor claridad los diferentes fenómenos que se presentan, especialmente con los múltiples tipos de sensores que forman parte integral de este sistema y que lo lleven a la práctica. El proceso de las prácticas con microcontroladores de Atmel hace posible obtener una plataforma de sencillo entendimiento y dar la noción al estudiante, para asociarse al universo de esta tecnología y así poder otorgarle al alumno suficiente conocimiento para elaborar proyectos con mayor complejidad y así desempeñar de mejor manera su carrera profesional.

Organización del informe

Capítulo 1. Análisis de la tecnología Atmel y prácticas de laboratorio.

Capítulo 2. Desarrollo de prácticas de laboratorio

Capítulo 3. Resultados y discusión de las prácticas de laboratorio

CAPÍTULO 1. ANÁLISIS DE LA TECNOLOGÍA ATMEL Y PRÁCTICAS DE LABORATORIO

Es amplia y diversa la bibliografía acerca de la tecnología Atmel, así como su implementación en la actualidad y los avances que ha tenido. En este capítulo se realiza una reseña bibliográfica acerca de los principales fabricantes de microcontroladores en el mundo. A través del estudio de los microcontroladores de Atmel el capítulo se enfoca en analizar el microcontrolador ATmega88PA. Se abordan también temas sobre equipamientos y dispositivos utilizados en tecnología Atmel. Se realiza una fundamentación de las prácticas de laboratorio. Se finaliza el capítulo arribando a conclusiones parciales.

1.1 ¿Qué es un microcontrolador?

Un microcontrolador es un chip único que está compuesto al menos por una unidad de procesamiento central (CPU por sus siglas en inglés), memoria no volátil, memoria volátil, un temporizador y una unidad de control de E/S, como se muestra en la figura 1.1 (Meroth, 2017).

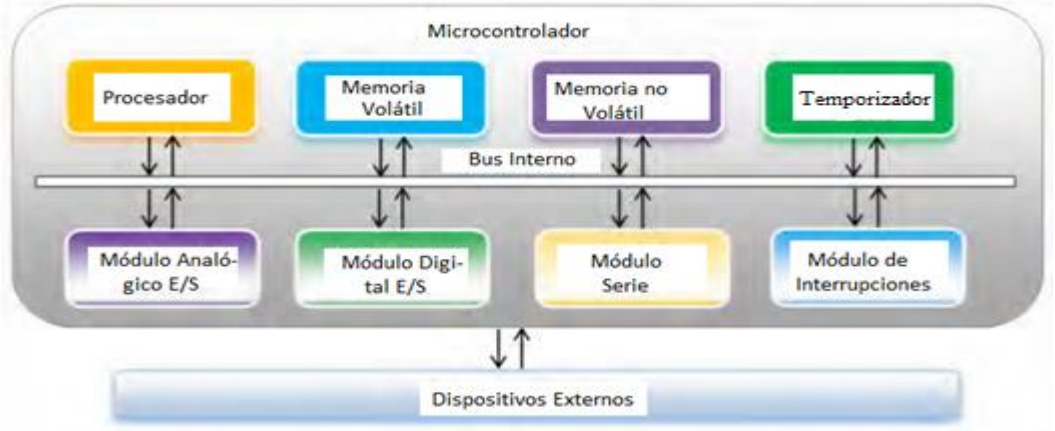


Figura 1.1. Representación de un microcontrolador.

Por su composición el microcontrolador es un dispositivo programable, capaz de realizar diferentes actividades, de control y comunicación entre diferentes dispositivos (Molina Tufiño, 2012).

Los microcontroladores poseen una memoria interna que almacena dos tipos de informaciones: instrucciones y datos. Las primeras corresponden al programa que se ejecuta, y los segundos son los datos que el usuario maneja. La programación puede ser escrita en lenguaje ensamblador u otro lenguaje para microcontroladores; sin embargo, para que el programa pueda ser grabado en la memoria del microcontrolador, debe ser codificado en sistema numérico hexadecimal que es finalmente el sistema que hace trabajar al microcontrolador cuando éste es alimentado con el voltaje adecuado y asociado a dispositivos analógicos para su funcionamiento. Cada microcontrolador varía su conjunto de instrucciones de acuerdo a su fabricante y modelo. De acuerdo al número de instrucciones que el microcontrolador maneja se le denomina de arquitectura RISC (acrónimo en inglés de Computador con Conjunto de Instrucciones Reducidas) o CISC (siglas en inglés para Computador con Conjunto de Instrucciones Complejas) (Petre Sora, 2018).

1.1.1 Fabricantes de microcontroladores

El primer microprocesador fue el Intel 4004 de 4bits, lanzado en 1971, seguido por el Intel 8008 y otros más capaces. Sin embargo, ambos procesadores requieren circuitos adicionales para implementar un sistema de trabajo, elevando el costo del sistema total (Villalobos, 2012).

Los ingenieros de Texas Instruments Gary Boone y Michael Cochran lograron crear el primer microcontrolador, TMS 1000, en 1971: fue comercializado en 1974. Combina memoria de solo lectura (ROM por sus siglas en inglés), memoria de acceso aleatorio (RAM por sus siglas en inglés), microprocesador y reloj en un chip y estaba destinada a los sistemas embebidos (Villalobos, 2012).

Debido en parte a la existencia del TMS 1000, Intel desarrolló un sistema de ordenador en un chip optimizado para aplicaciones de control, el Intel 8048, que comenzó a comercializarse en 1977. Combina memoria RAM y ROM y puede encontrarse en más de

mil millones de teclados compatibles con la computadora personal IBM[2] PC (figura 1.2), y otras numerosas aplicaciones. El presidente de Intel en aquel momento, Luke J. Valenter, declaró que el microcontrolador es uno de los productos más exitosos en la historia de la compañía, y amplió el presupuesto de la división en más de un 25% (Francisco., 2019).

Figura 1.2. IBM PC (modelo 5150).

La mayoría de los microcontroladores en ese momento tenían dos variantes. Unos tenían una memoria de solo lectura programable y borrable (EPROM por sus siglas en inglés) reprogramable, significativamente más caros que la variante memoria programable de solo lectura (PROM por sus siglas en inglés). Para borrar la EPROM se necesita exponer a la luz ultravioleta la tapa de cuarzo transparente. Los chips con todo opaco representaban un coste menor (Francisco., 2019).

En 1993, se lanza la EEPROM en los microcontroladores (comenzando con el Microchip PIC16x84). Esta memoria permite el borrado de forma eléctrica y rápida, sin necesidad de un paquete costoso como se requiere en EPROM, lo que facilita tanto la creación rápida de prototipos y la programación en el sistema. El mismo año, Atmel lanza el primer microcontrolador que utiliza memoria flash (figura 1.3). Otras compañías rápidamente siguieron el ejemplo, con los tipos de memoria (Microchip, 2018).

[2] Acrónimo en inglés para Corporación Internacional de Máquinas de Negocios

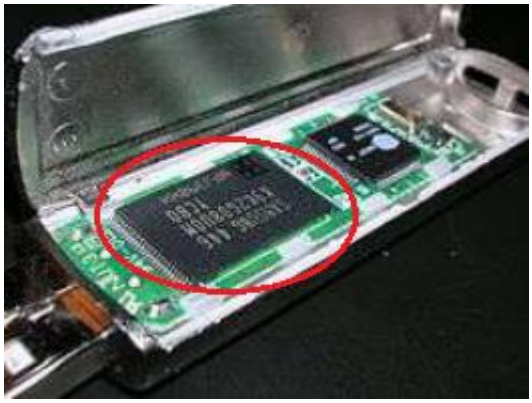


Figura 1.3. Memoria USB[3], el chip que se encuentra enmarcado a la izquierda es la memoria flash.

Dentro de estos tipos de microcontroladores, a menudo existen diversas configuraciones disponibles tales como tamaño de palabra 8, 16 y de 32 bits. El tamaño de palabra se refiere al tamaño de los números binarios que pueden ser manejados por el microcontrolador. Así mismo, los dispositivos de propósito general vienen en diferentes configuraciones de memoria y periféricos. Estos microcontroladores normalmente tienen un conjunto de características que serían útiles en una variedad de aplicaciones y se pueden diseñar en productos tales como electrodomésticos y productos de consumo (Petre Sora, 2018).

Existen muchos fabricantes de microcontroladores, entre los más importantes están: Microchip Technology, Atmel, Texas Instruments, Intel Corporation y National Semiconductor (Torres Alafita, 2004).

Microchip Technology es una empresa norteamericana, establecida en el estado de Arizona dedicada a la venta de componentes electrónicos, concretamente memorias, productos analógicos, radio frecuencia, microcontroladores, etc (Francisco., 2019).

La familia de microcontroladores de 8 bits, son llamados controlador de interfaz periférico (PIC por sus siglas en inglés) y la de 16 bits son las llamadas PIC24F, PIC24H, dsPIC30 y dsPIC33. También incluyen entre sus productos, microcontroladores de 32 bits, llamados PIC32. Todos presentan arquitectura Harvard (Microchip, 2018).

[3] Puerto bus universal en serie (USB por sus siglas en inglés)

Figura 1.4. Logo de Microchip Technology.

Familias de Microchip (Microchip, 2018).

- Arquitectura de 8 bits
  - PIC10: tienen 12 bits de palabra de programa y destacan por su bajo precio y número de pines escasos. Son potentes ya que pueden contener periféricos de reloj interno, conversor analógico digital (ADC por sus siglas en inglés), comparadores, interrupciones externas y, por lo tanto, son ideales para cuando el tamaño es escaso y la cantidad de pines no es demasiado grande.
  - PIC12: son más potentes que la familia PIC10 por tener más prestaciones. Tiene un bajo coste y un número abundante de periféricos.
  - PIC16: la familia es muy parecida a la PIC12, en cambio, dispone de más pines.
  - PIC18: es la familia más alta de Microchip, y una de las que se están usando más acorde al paso del tiempo, presenta una cantidad de pines y periféricos considerables, haciéndolos ideales para proyectos con envergadura y complejidad media-alta.
- Arquitectura de 16 bits
  - PIC24: el cual tiene un alto rendimiento y bajo costo, de él podemos encontrar dos subfamilias:
    - PIC24F: es la subfamilia que ofrece Microchip para aplicaciones de bajo coste, donde se encuentran ya con memorias flash hasta 128 kB y una velocidad de procesamiento de 16 millones de instrucciones por segundo (MIPS por sus siglas en inglés).
    - PIC24H: es la hermana mayor. Se utiliza para proyectos con un alto rendimiento y con una velocidad de procesamiento de 40 MIPS y memoria flash de programa hasta 256 kB.
  - DSPIC: dispositivos especializados en trabajar en el procesamiento digital de señales. Dos de sus subfamilias son:

    - DSPIC30: cuya aplicación es necesaria para proyectos de sistemas embebidos en tiempo real con una alimentación de 5 V a una velocidad de procesamiento de 30 MIPS.
    - DSPIC33F: se distingue de la subfamilia antes mencionada por tener una tensión de alimentación de 3.3 V a una velocidad de procesamiento de 40 MIPS. Además, dispone de más memoria de programa flash y de memoria RAM.
- Arquitectura de 32 bits
    - PIC32: diseñados para aplicaciones embebidas que requieran cantidades de memoria, procesamiento de la información y periféricos mayores.

National Semiconductor es una empresa estadounidense, fabricante de productos electrónicos, semiconductores, aunque también de forma muy específica fabrican microcontroladores. Los de 8 bits llamados COP8C y COP8S. Los de 16 bits se denominan CR16. Todos con arquitectura Harvard (Francisco., 2019).

Figura 1.5. Logo de National Semiconductor.

Familias de National Semiconductor (Villalobos, 2012).

- Arquitectura de 8 bits

Estos microcontroladores son los llamados COP8 y se dividen en tres familias.

    - COP8C: microcontroladores con 32 kB de memoria flash y 1kB de RAM. Se destacan por tener el periférico ADC de 10 bits.
    - COP8S: Son iguales que la familia COP8C, pero sin el periférico ADC.
    - COP8A: Es la gama alta de 8 bits, aunque poseen menos memoria de programa, tienen integrados más periféricos.
- Arquitectura de 16 bits
    - CR16: esta familia es la mayor de los microcontroladores de National y son de propósito general.

Texas Instruments está situada en Estados Unidos. Es líder en fabricación de semiconductores. Entre sus productos destacan los procesadores digitales de señales (DSP por sus siglas en inglés) y microcontroladores. Es normalmente conocida por sus siglas TI, donde su jerarquía de microcontroladores es: 16 bits y 32 bits, basados en arquitectura ARM7 (Barrientos Rojas, 2017).

Figura 1.6. Logo de Texas Instruments.

Familias de Texas Instruments (Torres Alafita, 2004).

- Arquitectura de 16 bits

Estos microcontroladores se destacan por ser de bajo consumo, con arquitectura RISC, y son los denominados MSP430.

Destacan varias subfamilias:

  - MSP430x1xx: son de propósito general y de bajo consumo, no tienen módulos de LCD y su memoria es del tipo flash-ROM.
  - MSP430F2xx: están basados en memoria flash con una rapidez de procesamiento de 16 MIPS con una cantidad de periféricos notables para proyectos complejos en necesidad de memoria y periféricos.
  - MSP430x3xx: se basan en memoria no volátil de solo lectura programable una sola vez (OTP por sus siglas en inglés), están orientados para procesos industriales, su velocidad de procesamiento es de 8 MIPS con una cantidad de memoria de programa considerable.
  - MSP430x4xx: es la categoría alta de Texas Instruments, con una cantidad de memoria alta al igual que su velocidad de procesamiento. Con módulo LCD y tecnología de memoria flash.

- Arquitectura de 32 bits

En esta arquitectura, disponemos de los que son mundialmente conocidos, como los procesadores de señal.

  - TMS320F283xx: controladores en punto flotante.
  - TMS320F281xx: capaces de trabajar a una velocidad de procesamiento de 150 MIPS.
  - TMS320F280xx: capaces de trabajar a una velocidad de procesamiento de 100 MIPS.
- Arquitectura ARM

Aunque son de 32 bits están basados en el núcleo ARM7.

  - TMS470: dedicados para procesos industriales.
  - SM470: dedicados a la automoción.

Toshiba es una empresa japonesa, fabricante de productos electrónicos de consumo e industriales, así como semiconductores. Ofrece al usuario una gama amplia de microcontroladores de 8 y 16 bits con tecnología CISC y microcontroladores de 32 bits de arquitectura RISC (Villalobos, 2012).

Figura 1.7. Logo de Toshiba.

Familias de Toshiba (Torres Alafita, 2004)

- Arquitectura de 8 bits

Toshiba emplea en esta familia de 8 bits un gran número de microcontroladores con varios periféricos incorporados para reducir al máximo los componentes externos. En toda su familia se incluye puertos seriales y temporizadores tanto de 8 como de 16 bits.

Se pueden seleccionar microcontroladores tanto de propósito general como especializados en bus de red de área del controlador (CAN por sus siglas en inglés), controlador LCD y control de motores.

- Arquitectura de 16 bits

Es una familia mejorada orientada a los procesos industriales, aunque podemos encontrar dispositivos de propósito general, control de motores y control LCD. Aquí se mejora la cantidad de memoria de programa y de datos, así como los periféricos integrados.

- Arquitectura de 32 bits

Es la categoría grande de Toshiba, con una cantidad de periféricos integrados, así como memoria de datos y de programa bastante considerable, además está equipada con memoria nano flash.

Intel Corporation es el mayor fabricante de circuitos integrados del mundo. La compañía estadounidense es la creadora de la serie de procesadores x86, los procesadores más comúnmente encontrados en la mayoría de las computadoras personales. Intel fue fundada el 18 de julio de 1968 como Integrated Electronics Corporation (Barrientos Rojas, 2017).

Ejemplos más cercanos son el empleo de la RDRAM (es un tipo de memoria sincrónica) de los Módulos de Memoria en Línea Rambus (RIMM por sus siglas en inglés) y el Slot 1 (ranura de expansión) en los Intel Pentium II / Intel Pentium III, medidas tomadas para afianzar el dominio del mercado a golpe de patente, y que se acabaron volviendo en su contra al forzar a sus competidores a innovar y abaratar costes, logrando la compañía Micro Dispositivos Avanzados (AMD por sus siglas en inglés) llevar a buen puerto el primer procesador de 64 bits de la x86-64 que además mantenía la compatibilidad x86 (Barrientos Rojas, 2017).

Figura 1.8. Logo de Intel Corporation.

Familias de Intel (Villalobos, 2012).

- Arquitectura de 8 bits.

Los componentes de la familia 8051 encuentran aplicaciones que van desde el control de máquinas industriales y de instrumentación hasta el control automotriz. Los dispositivos de

la serie pueden obtenerse en versiones con ROM o EPROM internas o, solamente, con la CPU. Con excepción de la 83C751, todos los dispositivos de esta familia pueden manejar hasta 64 bytes, tanto de programa como de memoria de datos (Francisco., 2019).

Atmel es una empresa norteamericana. Sus productos se basan en todo lo relacionado a los semiconductores, memorias y dispositivos lógicos programables. Posee en fabricación, derivados del famoso 8051 y microcontroladores con arquitecturas propias de las familias AVR (*Advanced Virtual* RISC) y AVR32 (Technology, 2017a).

Figura 1.9. Logo de Atmel.

1.2 Tecnología Atmel

Atmel es una compañía de microcontroladores fundada en 1984 por George Perlegos. Perlegos había trabajado en el grupo de desarrollo de memoria de Intel en los años ’70 y era cofundador de una empresa de semiconductores fundada en 1981 llamada Seeq Technology para la producción de memoria EPROM. Atmel operó inicialmente como una compañía casera, usando materiales de Sanyo o General Instruments. Los primeros productos de memoria de la misma utilizaban menos energía que sus competidores atrayendo a clientes como Motorola, Nokia y Ericsson. En 1987 Intel demandó a Atmel por violación de patentes. En lugar de luchar contra la reivindicación de la patente la compañía rediseñó sus productos y utilizó diferentes patentes de propiedad intelectual. Sus componentes tenían un mejor rendimiento y menor consumo de energía (Tapia Moraga, 2012).

En sus inicios Atmel era una compañía de semiconductores. Actualmente en su línea de productos se incluyen microcontroladores, dispositivos de radio frecuencia, EEPROM y muchas otras. Además, entró en el negocio de memoria flash que Intel había protagonizado (Tapia Moraga, 2012).

Sus productos cogieron una gran aceptación en una gran cantidad de países y ciudades importantes del mundo, ya que brindan servicio a mercados de computadores, electrónica de

consumo, electrónica industrial, comunicaciones, automotriz, redes, militar, equipos médicos y aeroespacial. Además, es una industria líder en sistemas seguros (Tapia Moraga, 2012).

Es la compañía encargada de fabricar los microcontroladores de la familia AVR. Esta familia empleaba una nueva tecnología que proporcionaba todos los beneficios habituales de arquitectura RISC y memoria flash reprogramable eléctricamente. AVR compite con varias familias de microcontroladores bien establecidas en el mercado, tales como 8051 de Intel, 68HC11 de Motorola y PIC de Microchip. La firma también produce y vende varios subproductos de la popular familia 8051 con la diferencia de que están basados en la memoria flash (Vega Rosero, 2012).

Adicionalmente, Atmel también proporciona en línea, el software que permite editar, ensamblar y simular el código fuente. Una vez ensamblado y depurado, se debe transferir a la memoria flash del micro-controlador (Vega Rosero, 2012).

En 2016 la compañía fue adquirida por Microchip Technology. Entre sus principales competidores se encuentra STMicroelectronics, Texas Instruments, Freescale (ahora NXP Semiconductors), Analog Devices y Microchip Technology (Tapia Moraga, 2012).

1.2.1 Familias de microcontroladores que desarrolla Atmel

La familia AVR se basa en procesadores con núcleos RISC y arquitectura Harvard, estas son algunas de las subfamilias en las que se dividen sus dispositivos (Francisco., 2019):

- Automotive AVR: su principal característica es que poseen periféricos integrados tales como ADC de 10 bits, bus CAN y modulación por ancho de pulsos (PWM por sus siglas en inglés).
- AVR Z-Link: se especializan en trabajos basados en la tecnología ZigBee (es el nombre de la especificación de un conjunto de protocolos de alto nivel de comunicación para su utilización con radiofusión digital de bajo consumo, basada en el estándar del Instituto de Ingeniería Eléctrica y Electrónica (IEEE por sus siglas en inglés) de redes inalámbricas de área personal (WPAN por sus siglas en inglés).
- CAN AVR: poseen más de un canal del bus CAN, por lo tanto, serán ideales para proyectos que requieran controlar y manejar varios dispositivos de bus CAN.

- LCD AVR: creados para el manejo de segmentos de LCD, oscilan del 4x25 Segment LCD Driver al 4x40 Segment LCD Driver.
- Lighting AVR: diseñados para controlar lámparas y motores, lo cual significa que sus principales ventajas en periféricos son: varios canales del ADC, varios canales de PWM.
- megaAVR: es una familia de microcontroladores estándar de Atmel, caracterizada por su bajo costo.
- Smart Battery AVR: dedicado a dispositivos que requieren baterías, ya que su consumo de batería es muy bajo, ideal para proyectos móviles o portátiles.
- tinyAVR: igual que megaAVR es una familia estándar, cuya cualidad es su tamaño reducido y número de pines.
- USB AVR: se especializa en controlar y manejar el USB.

1.2.2 Análisis detallado de algunas de las familias de Atmel, microcontroladores de 8 y 16 bits.

- Familia megaAVR
  - Características:
  - CPU de 8/16 bits.
  - CPU con 32 registros de propósito general.
  - Conectividad USB, CAN.
  - Memoria flash entre 4 y 256 kB.
  - De 28 a 100 pines.
  - Comunicación SPI, USART.
  - Memoria SRAM[4] de hasta 16 kB.
  - CPU de hasta 32 MHz.
  - Conversores ADC y conversor digital-analógico (DAC por sus siglas en inglés).
  - Controladores LCD.

[4] Memoria estática de acceso aleatorio (SRAM por sus siglas en inglés)

Los microcontroladores que pertenecen a megaAVR son para aplicaciones de propósito general, junto con el manejo de LCD e iluminación. En el Anexo I, se muestran los microcontroladores fabricados de esta familia (Francisco., 2019).

- Familia tinyAVR
  - Características:
  - CPU de 8/16 bits.
  - Memoria flash entre 0.5 y 8 kB.
  - De 6 a 32 pines.
  - Operación desde 0.7 V.
  - CPU de hasta 20 MHz.
  - Conversores ADC.

Los microcontroladores que integran tinyAVR son para aplicaciones de propósito general donde el espacio es reducido. En el Anexo II se menciona su evolución de fabricación (Francisco., 2019).

1.2.3 Generalidades de los microcontroladores ATmega

Una de las familias que pertenecen al gran desarrollo de Atmel es AVR, dentro de ellas hay varias subfamilias como megaAVR la cual está integrada por los microcontroladores ATmega. Esta subfamilia está encargada del diseño y elaboración de unidades de procesamiento muy versátiles y con grandes capacidades de funcionamiento. Sus parámetros básicos son: arquitectura Harvard, reloj interno, escalabilidad, manejo de señales analógicas y temporizadores e interrupciones externas (Tapia Moraga, 2012).

Que contengan arquitectura Harvard, significa que tiene el bus de datos separado del bus de instrucciones lo cual le da mayor velocidad de procesamiento. Además, algunos de ellos poseen un oscilador y reloj interno, y pueden trabajar también con un oscilador externo. El mismo puede ser utilizado como reloj del sistema, si se desea se puede utilizar una fuente externa de sincronismo (Microchip, 2018).

En los Atmega también se destacan sus grandes velocidades de procesamiento: 1 MIPS, por 1MHz debido la arquitectura RISC que les permite ejecutar una instrucción por ciclo y a sus 32 registros para múltiples propósitos (Parra Sánchez, 2016).

1.2.4 Comparaciones de microcontroladores de Atmel con otras firmas.

En la Tabla 1.1 se muestran las características más notables en cuanto a la cantidad de memoria disponible y la velocidad de procesamiento, de microcontroladores similares en capacidades, tanto de Microchip como de Atmel (Tapia Moraga, 2012).

Tabla 1.1: Comparación entre el ATmega328p y el PIC16F877.

Características	PIC16F877	ATmega328p
Número de instrucciones disponibles	35	131
Cantidad máxima de pines de I/O	23	23
Velocidad de Procesamiento	5 MIPS a 20 MHz	16 MIPS a 16 MHz
Capacidad de memoria flash (programa)	8 kB	32 kB
Capacidad de memoria RAM (datos)	368 Bytes	2 kB
Capacidad de EEPROM (datos)	256 Bytes	1 kB

Con la comparación de ambos tipos de microcontroladores, podemos observar que el micro perteneciente a la familia AVR posee mejores características de funcionamiento que el de la familia PIC, se aprecia que los microcontroladores de ATmega ofrecen más ventaja (Tapia Moraga, 2012).

En la tesis elaborada por (Martínez, 2011) se evalúan dos microcontroladores para escoger uno de ellos, teniendo en cuenta la duración de las instrucciones en ciclos de reloj. El primero es un ATmega88PA y el segundo un PIC18F2550. Se llega a la conclusión de que, en el PIC, cada instrucción demora como mínimo 4 ciclos de reloj, mientras que en el ATmega88PA cada instrucción demora como mínimo 1 ciclo. La Tabla 1.2 presenta una comparación entre instrucciones básicas de ambos microcontroladores.

Tabla 1.2: Comparación de Instrucciones Básicas de ATmega88PA y PIC18F2550.

Instrucción ATmega88PA	Ciclos de reloj	Instrucción PIC18F2550	Ciclos de reloj

LDI	1	MOVLW	4
CLR	1	CLRF	4
ADD	1	ADDWF	4
DEC	1	DECF	4
BRNE	1 o 2	BNZ	4 u 8
PUSH	2	PUSH	4
POP	2	POP	4
LPM	3	TBLRD	8
RCALL	3	RCALL	8
RET	4	RETURN	8

A partir de esta tabla se infiere que para que un PIC18F2550 pueda ejecutar un programa a una velocidad aproximadamente igual que un ATMEGA88PA, es necesario que funcione a una frecuencia 4 veces mayor (Martínez, 2011).

El siguiente criterio de selección fue la potencia consumida. Para ello se evalúa el consumo de corriente en estado activo (Tabla 1.3). Los parámetros de mayor influencia en el consumo de corriente son la frecuencia de operación y el voltaje de alimentación. Se realiza la evaluación con alimentación de 5 voltios ya que a este nivel es posible utilizar el microcontrolador PIC a su máxima frecuencia (48 MHz) (Martínez, 2011).

Tabla 1.3: Consumo de Corriente de ATmega88PA y PIC18F2550.

ATmega88PA		PIC18F2550	
Vcc = 5V		Vcc = 5V	
Frecuencia (MHz)	Corriente (mA)	Frecuencia (MHz)	Corriente (mA)

0.25	0.2	1	1.1
1	0.8	4	2.5
10	5	40	21
12	5.8	48	25

Con las comparaciones y evaluaciones que se hicieron en el trabajo se llegó a la conclusión que el ATmega88PA tiene mayor velocidad de ejecución y eficiencia energética que el microcontrolador PIC18F2550 (Martínez, 2011).

1.2.5 Productos que fabrica y desarrolla Atmel

En la figura 1.10 se ve la versión del microcontrolador 8051, el AT89C51 fabricado por Atmel. Presenta una CPU de 8 bits, memoria de programa flash 4 kB, 128 Bytes de RAM, reloj hasta 24 MHz, 32 pines de entradas/salidas (E/S) disponibles, 2 temporizadores/contadores de 16 bits, 6 fuentes de interrupción, comunicación Transmisor-Receptor Universal Asincrónico (UART por sus siglas en inglés) *full dúplex* y modos de ahorro de energía.

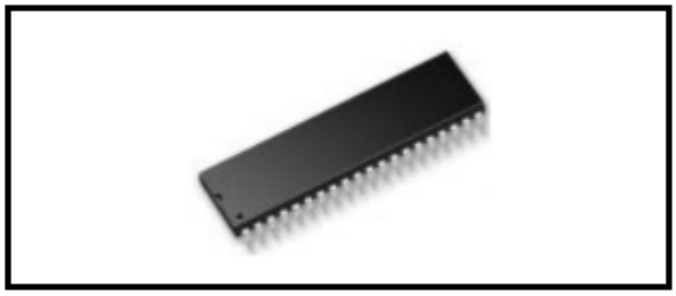


Figura 1.10. Microcontrolador AT89C51.

Las siguientes prestaciones pertenecen a la figura 1.11: microcontrolador de 8 bits, 23 pines de E/S (entradas y salidas) disponibles, memoria de programa flash 16 kB, 1 kB de SRAM, EEPROM de datos 512 Bytes, ADC de 10 bits y 8 canales, 3 temporizadores/comparadores/contadores, 6 canales de PWM, Transmisor-Receptor Universal Sincrónico y Asincrónico (USART por sus siglas en inglés), Bus SPI (siglas en inglés para bus de periféricos con interfaz serie), circuito inter-integrado (I^2C por sus siglas

en inglés), comparador analógico y montaje superficial Encapsulado Plano Cuádruple Delgado (TQFP por sus siglas en inglés).

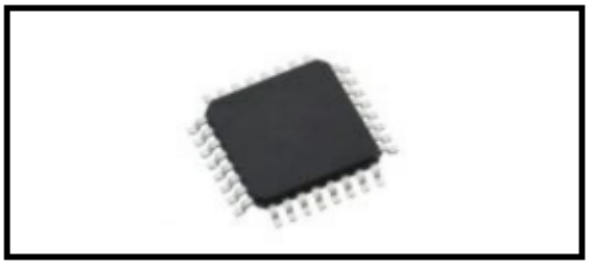


Figura 1.11. Microcontrolador Atmega168A.

En la figura 1.12 se aprecia un programador y depurador por USB original de Atmel que permite programar en circuito y depura todos los microcontroladores AVR de 8 y 32 bits.

Figura 1.12. Atmel-ICE kit completo.

Se ve en la figura 1.13 un microcontrolador de 8 bits, 18 pines de E/S disponibles, memoria de programa flash de 2 kB, SRAM de 128 Bytes, EEPROM de datos 128 Bytes, 2 timers/comparadores/contadores, 4 canales de PWM, comunicación USART, SPI y comparador análogo.

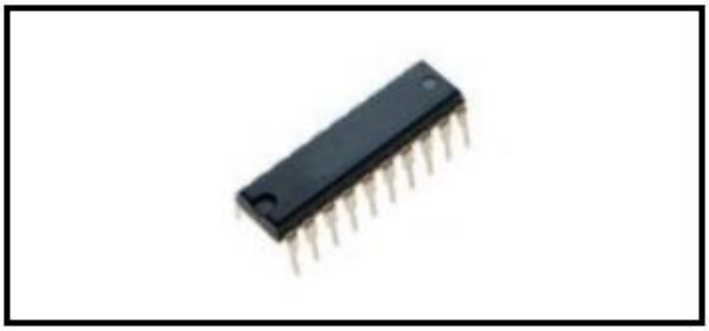


Figura 1.13. Microcontrolador ATtiny2313A.

1.3 Microcontrolador ATmega88PA

Un microcontrolador ATmega88PA combina un rico conjunto de instrucciones con 32 registros de trabajo de propósito general. Los 32 registros están directamente conectados a la unidad aritmético/lógica (ALU por sus siglas en inglés), lo que permite dos registros independientes para acceder en una sola instrucción ejecutada en un ciclo de reloj (Robayo Alcocer, 2014).

El ATmega88PA proporciona las siguientes características: 8K bytes de flash programable en el sistema con capacidades de lectura mientras escribe, EEPROM de 512 bytes, SRAM de 1K bytes, 23 líneas de E/S de propósito general, 32 registros de trabajo de uso general, tres temporizadores/contadores flexibles con modos de comparación, interrupciones internas y externas, un USART programable en serie, un cable de 2 hilos orientado a bytes, interfaz serie, un puerto serie SPI, un ADC de 6 canales, un temporizador de vigilancia programable con oscilador interno y cinco selecciones de software. El modo inactivo detiene la CPU mientras permite que continúen trabajando la SRAM, el temporizador, los contadores, USART, la interfaz serie de 2 hilos, el puerto SPI y el sistema de interrupción para continuar el racionamiento. El modo de apagado guarda el contenido del registro, pero congela el oscilador, deshabilitando todas las demás funciones de chip hasta la próxima interrupción o reinicio de hardware. En el modo de ahorro de energía, el temporizador asincrónico continúa ejecutándose, lo que permite al usuario mantener una base del temporizador mientras el resto del dispositivo está durmiendo. El modo de reducción de ruido del ADC detiene la CPU y todos los módulos de E/S excepto el temporizador asincrónico y el ADC, para minimizar el

ruido de conmutación durante las conversiones del ADC (Robayo Alcocer, 2014), (Atmel, 2008).

En modo de espera, el oscilador de cristal/resonador está funcionando mientras el resto del dispositivo está durmiendo. Esto permite un arranque muy rápido combinado con un bajo consumo de energía (Atmel, 2008).

El dispositivo se fabrica utilizando la tecnología de memoria no volátil de alta densidad de Atmel. El programa de arranque puede usar cualquier interfaz para descargar el programa de aplicación en la memoria flash de la aplicación. El software en la sección *Boot* flash continuará ejecutándose mientras se actualiza la sección de *Application* flash, proporcionando una verdadera operación de lectura-escritura, al combinar una CPU RISC de 8 bits con un flash autoprogramable en el sistema en un chip monolítico, el Atmel ATmega88PA es un potente microcontrolador que proporciona una gran flexibilidad y una rentable solución económica para muchas aplicaciones de control integradas (Microchip, 2018).

ATmega88PA de la familia AVR es compatible con un conjunto completo de herramientas de desarrollo de programas y sistemas, incluyendo: compiladores de C, ensambladores, depuradores/simuladores de programas, emuladores en circuito, y kits de evaluación (Atmel, 2008).

1.3.1 Configuración de pines

Los microcontroladores responden a una configuración de pines específica. La figura 1.13 muestra cómo están configurados los pines en un Atmega88PA y en el Anexo IV se muestra una tabla con el significado de los colores de los pines de dicha figura (Petre Sora, 2018).

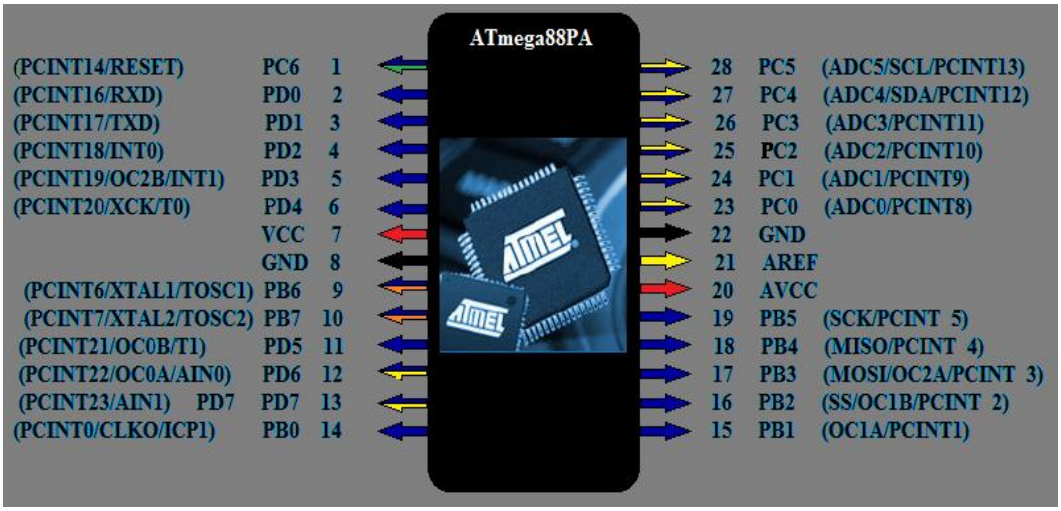


Figura 1.13. Configuración de pines del microcontrolador ATmega88PA.

Los pines de cada microcontrolador responden a elementos lógicos los cuales están encargados de realizar las diferentes funciones del mismo. A continuación, describimos los elementos lógicos y sus pines correspondientes (Petre Sora, 2018), (Atmel, 2008).

- **VCC**

Pin de tensión de alimentación, pin 7.

- **GND**

Tierra, pines 8 y 22.

- **Puerto B:**

Es un puerto de E/S bidireccional de 8 bits con resistencias internas (seleccionadas para cada pin). Los búferes de salida del puerto B tienen características de accionamiento simétrico con capacidad de fuente y sumidero alto. Los pines del puerto B que se extraen externamente en bajo como entradas, generarán corriente si se activan las resistencias, estas características son comunes para los tres puertos B, C, D con la excepción de la cantidad de pines del puerto C que no son 8 sino 7. También los pines que corresponden a cada puerto son diferentes, los pines que utiliza el puerto B son: 9, 10, 14, 15, 16, 17, 18, 19.

- **Puerto C**

Los pines que lo conforman son: del 23 al 28.

  - **PC6**

Si el fusible está programado, PC6 se usa como un pin de E/S. Las características eléctricas de PC6 difiere de los otros pines del puerto C. Si el fusible no está programado, PC6 se usa como entrada de reinicio. Un nivel bajo en este pin por más tiempo que la duración mínima del pulso generará un reinicio, incluso si el reloj no está funcionando. Que haya pulsos más cortos no garantiza que genere un reinicio. Este elemento lógico pertenece al puerto C y está integrado solamente por el pin 1.

- **Puerto D**

Está compuesto por los pines del 2 al 6, y 11 al 13.

- **AVCC**

Es el pin de voltaje de suministro para el ADC. Debe ser externamente conectado a VCC, incluso si no se utiliza el ADC. Si se utiliza, debe conectarse a VCC a través de un filtro paso bajo. Está conformado por el pin 20.

- **AREF**

Pin de referencia analógico para el ADC, pin 21.

En el Anexo III se explica el significado de los colores de los pines de la figura 1.13.

1.3.2 Arquitectura

Para maximizar el rendimiento, la familia AVR utiliza una arquitectura Harvard, con memorias y buses para programa y datos. Las instrucciones en la memoria del programa se ejecutan con un canal de un solo nivel. Mientras se ejecuta una instrucción, la siguiente instrucción se obtiene previamente de la memoria de programa. Este concepto permite ejecutar instrucciones en cada ciclo de reloj. La memoria de programa es memoria flash reprogramable en el sistema (Petre Sora, 2018).

Con la arquitectura "Von Neumann", la memoria del programa y la memoria de trabajo están activadas, se conecta a un bus, por lo tanto, se accede a los programas y a los datos secuencialmente. Los controladores integrados, por otro lado, a menudo se mantienen con arquitectura Harvard, los procesadores de la familia AVR fueron desarrollados en la Universidad de Harvard en Boston. En la figura 1.14 se muestran ambas arquitecturas (Meroth, 2017).

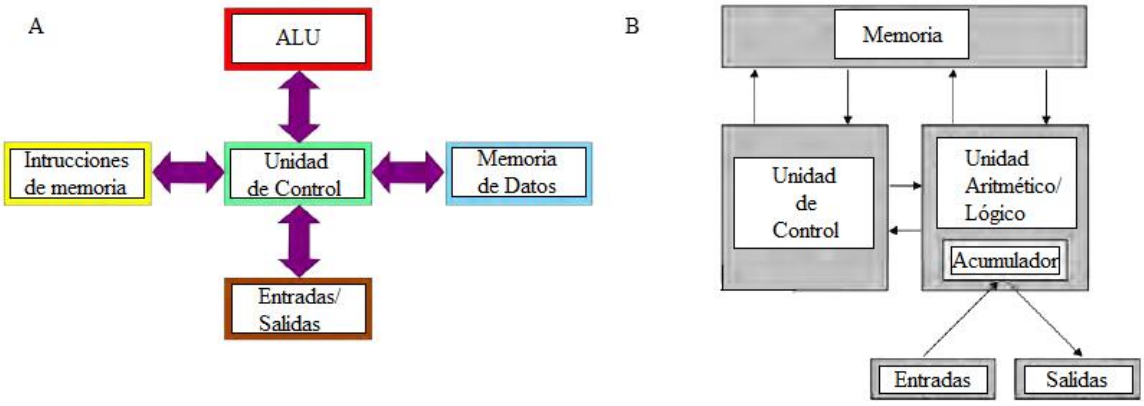


Figura 1.14. A. Arquitectura Harvard, B. Von Neumann

La arquitectura exige una separación estricta entre programa, datos, unidad aritmética y periféricos. La ventaja de este concepto es que el software es compacto, en uno se almacena la memoria y los datos están en la memoria de trabajo o en otras memorias no volátiles (EEPROM). Porque durante el plazo el programa no se cambia de todos modos, la flexibilidad de la arquitectura Neumann no es necesaria. El acceso al programa y a los datos es mucho más rápido y más eficiente que con la arquitectura Von Neumann (Petre Sora, 2018).

1.3.3 Reloj

Los procesadores de la familia AVR tienen una serie de señales de reloj que son generadas por un reloj central (Petre Sora, 2018). Las principales fuentes para el reloj son:

Un oscilador interno que se sincroniza a 8 MHz sin circuitos externos y otro con 128kHz.

Un oscilador con un resonador externo (cuarzo o cerámica) conectados al puerto B por PB6 y PB7 pines 9 y 10 respectivamente en las funciones XTAL1 y XTAL2 y para la familia ATmega Se pueden sincronizar hasta 20 MHz.

Un oscilador externo que envía una señal de onda cuadrada al oscilador XTAL1 del procesador, XTAL2 se puede utilizar como un pin E/S.

Los escenarios de uso típicos son:

Uso solo del oscilador interno.

Uso de un cuarzo externo con la frecuencia de reloj completa.

Uso del oscilador interno con frecuencia de reloj completa y conexión de un cuarzo.

1.3.4 Reset

Restablecer el procesador (RESET) hace que el contador del programa se restablezca y muestre la dirección 0 en la memoria del programa. En la dirección 0 hay un comando de salto en la ubicación del programa principal main (), es decir, el programa comienza allí (Microchip, 2018).

El procesador se puede restablecer externamente a través de varios mecanismos:

- Cuando se enciende (reinicio de encendido), el condensador sin carga se forma en el RESET pin uno (PC6) perteneciente al puerto C. Esto restablecerá el pin de restablecimiento 1 brevemente a 0.
- Un botón RESET en el pin RESET provoca un reinicio manual.
- El ATmega88PA solo se puede programar si el procesador está en modo RESET, por lo que el dispositivo de programación abre la línea RESET.

1.3.5 Almacenamiento

La figura 1.15 muestra los tres tipos de almacenamiento del ATmega88PA. En a se muestra la memoria de programa, donde en 0x0000 empieza la sección para la aplicación de la memoria flash y de 0x0FFF a 0x1FFF está destinado a la sección de arranque de la misma. En b se aprecia como esta segmentado el almacenamiento de datos, registros, registros de E/S, registros externos de E/S y SRAM interna. En c se observa la EEPROM con una capacidad de 512 Bytes (Microchip, 2018).

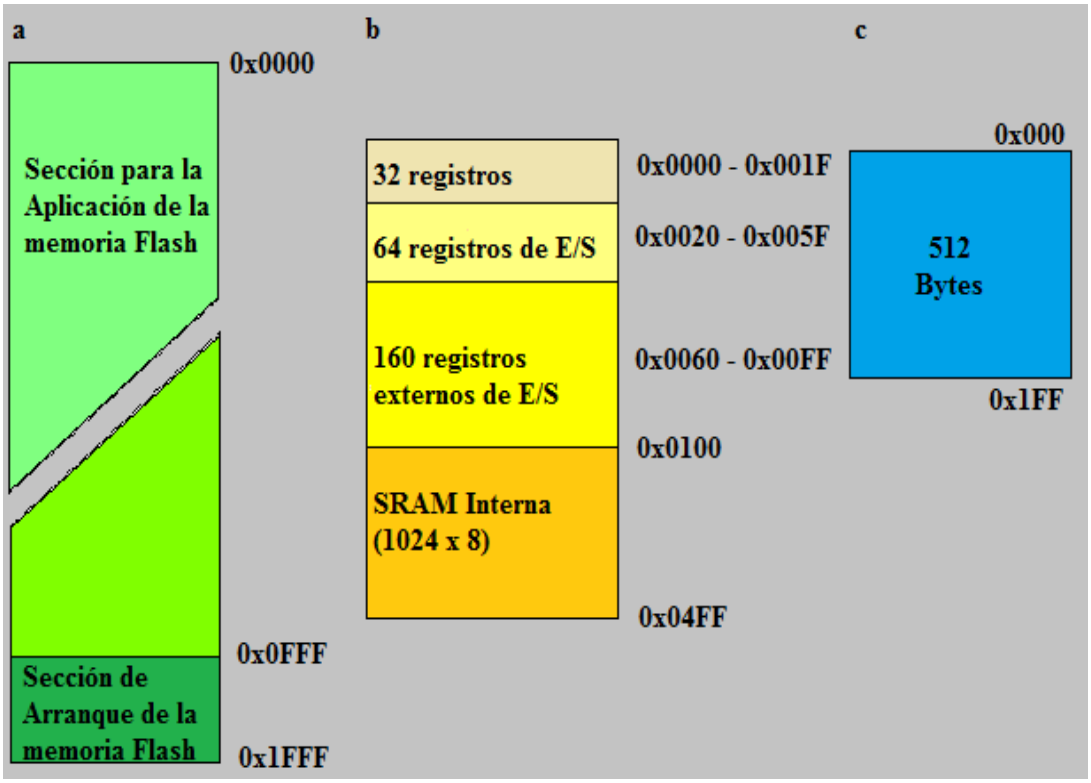


Figura 1.15. Tipos de almacenamiento del microcontrolador ATmega88PA.

1.3.6 Suministro

Los procesadores de la familia AVR generalmente se usan con 1.8V a 5V. Con una frecuencia de cuarzo más alta, generalmente se requiere un voltaje más alto. Si el voltaje cae, debido a una batería vacía, pueden ocurrir inestabilidades. Por lo tanto, muchos microcontroladores tienen una llamada detección de oscurecimiento que restablece el procesador por debajo de un nivel de voltaje de suministro ajustable.

Al medir la tensión de alimentación con una fuente interna independiente de la alimentación de referencia, por ejemplo, las operaciones de escritura se pueden completar en la EEPROM antes de que comience un restablecimiento de caída de voltaje. En circuitos reales, por ejemplo, se deben usar las baterías para asegurar, en caso de caída de la tensión de alimentación (Microchip, 2018).

1.4 Fundamentación metodológica de las prácticas de laboratorio

A propuesta de John Locke se introduce la práctica de laboratorio en la educación, hace casi trescientos años. Al entender la necesidad de realización de trabajos prácticos en la formación de los alumnos. Ya a finales del siglo XIX formaba parte integral del currículo de las ciencias en Inglaterra y Estados Unidos (Gee, 1992), (Layton, 1990), (Lock, 1988). Desde entonces, se ha mantenido una fe inmensa en la tradición que asume la gran importancia del trabajo práctico para la enseñanza de las ciencias (Barberá, 1996).

Se han desarrollado investigaciones en las últimas décadas sobre las prácticas de laboratorio. Esto ha generado un amplio consenso en torno a la orientación del trabajo experimental como una actividad investigativa, lo que juega un papel primordial en la familiarización de los estudiantes con la metodología científica (Urrea, 2013).

En varias investigaciones se afirma que enseñar efectuando investigaciones y prácticas ofrece al personal docente la oportunidad de que sus alumnos desarrollen aptitudes para enriquecer el conocimiento técnico y científico. Lo anterior apoya la importancia de la construcción y documentación de un marco teórico para la enseñanza y el aprendizaje de la ciencia y la tecnología a través de una docencia basada en el trabajo práctico y la investigación (Ortega Díaz, 2015).

La aplicación de formas alternativas de aprendizaje es una opción frente al desarrollo de la metodología de enseñanza tradicional, en la que prevalece la clase magistral (Scoles, 2012). El empleo de métodos menos pasivos, hace que los alumnos perciban el laboratorio como un lugar donde están activos (Torres, 2013).

En resumen, las prácticas de laboratorio fortalecen la construcción en el estudiante de cierta visión sobre la ciencia. Con esta visión ellos pueden entender que acceder a la ciencia no es imposible y, además, que la ciencia no es infalible. La ciencia depende de otros factores o intereses (sociales, políticos, económicos y culturales) (Hodson, 1994). La actividad experimental hace mucho más que apoyar las clases teóricas de cualquier área del conocimiento; su papel es importante en cuanto despierta, desarrolla y consolida la curiosidad de los estudiantes, ayudándolos a resolver problemas y a explicar y comprender los fenómenos con los cuales interactúan en su cotidianidad (López, 2012).

En la ingeniería, un laboratorio bien diseñado es una valiosa herramienta que ayuda a solidificar la enseñanza. En ellos los estudiantes pueden verificar un modelo, validar y limitar suposiciones y predecir rendimientos (Guadalupe, 2006). Las prácticas de laboratorio deben favorecer el análisis de resultados por parte de los estudiantes. La elaboración de un informe final, debe especificar claramente el problema planteado, las hipótesis emitidas, las variables que se tuvieron en cuenta, el diseño experimental realizado, los resultados obtenidos y las conclusiones. Finalmente se debe producir una evaluación coherente con todo el proceso de resolución de problemas y con criterios referidos al trabajo científico y al aprendizaje profundo (Hodson, 1996), (González, 1994), (Dourado, 2006).

Varias teorías apuntan a que las prácticas de laboratorio deben estar estructuradas de forma tal que se cumpla: objetivos, tarea preliminar, técnica operatoria, y conclusiones (Santín, 2011).

1.5 Conclusiones Parciales

Con el análisis de la tecnología Atmel y de otras firmas se puede apreciar que cada familia de microcontroladores está diseñada para un determinado requerimiento con una determinada eficiencia. No obstante, su empleo es indispensable en la industria actual y por ello, también el conocer los diversos dispositivos ofrecidos por los fabricantes para su aplicación.

Todos los microcontroladores tienen estructuras y arquitecturas básicas similares, lo que hace que sea muy sencillo utilizar cualquiera de los productos de los diferentes fabricantes, dando la posibilidad de migrar entre ellos según la necesidad.

Con el análisis de los fundamentos teóricos de las prácticas de laboratorio se aprecia la repercusión de gran importancia que tuvieron las mismas en el transcurso del tiempo, dando lugar a la selección de la estructura a presentar en este trabajo.

CAPÍTULO 2. DESARROLLO DE PRÁCTICAS DE LABORATORIO

El capítulo se centra en dar solución al trabajo de investigación, para ello se analizó el hardware y el software empleados para el desarrollo de las prácticas. Se comprobaron las conexiones entre uno y otro para el desarrollo de las mismas tributando a un empleo eficiente para el aprendizaje y desarrollo por parte de los estudiantes.

2.1 Composición del módulo µECU

Para la realización de las prácticas se cuenta con un módulo adquirido a través de la colaboración con una universidad alemana. A continuación, se presentan los principales componentes que se utilizan del módulo para la realización de estas prácticas de laboratorio y su conexión con el microcontrolador ATmega88PA.

- Microcontrolador ATmega88PA es el corazón de este módulo ya que ejecuta los programas realizados. Se ve resaltado con un óvalo de color blanco en la parte superior de la figura 2.1.
- Cuatro leds, de ellos dos solamente led1 y led2 son manejados por el microcontrolador. El led1 es de color rojo y está conectado al puerto B del microcontrolador ATmega88PA por el pin 16 (PB2). El led2 es de color verde y está conectado al puerto D por el pin 12 (PD6). Los dos leds se aprecian en un óvalo de color azul en la parte inferior de la figura 2.1.
- Cuatro botones de tacto S1, S2, S3, S4. Todos están conectados al puerto D. S1 se conecta al microcontrolador ATmega88PA por el pin 11 (PD5), S2 por el pin 6 (PD4), S3 por el pin 5 (PD3) y S4 por el pin 4 (PD2). Los botones cuentan con protección de polaridad inversa y se pueden ver en un óvalo de color amarillo en la parte inferior de la figura 2.1.

- Un conector USB capaz de establecer la comunicación con equipos de cómputo. Se puede ver en la parte superior de la figura 2.1 enmarcado en un óvalo de color rojo.

Figura 2.1. Microcontrolador ATmega88PA, led 1 (izquierda), led2 (derecha), botones S1, S2, S3, S4 (de izquierda a derecha) y conector USB.

- Una interfaz serial FT232 BL, (se aprecia en la figura 2.2 enmarcada en un óvalo de color morado), encargado de la comunicación serie del dispositivo.
- Dos sensores, un termistor para medir temperatura y una fotorresistencia para medir la luminosidad, ambos están conectados al puerto C, el sensor de temperatura por el pin 28 (ADC5) y el sensor de luz por el pin 27 (ADC4). Se pueden ver en la parte inferior de la figura 2.2 en un óvalo de color verde.
- Un *LCD-Display* 162C utilizado para la visualización en el módulo. Se conecta por los puertos B y C. El pin 4 (RS) de la pantalla se conecta al pin 14 (PB0) del microcontrolador ATmega88PA. El pin 6 (EN) se conecta al pin 15 (PB1). Los pines 11, 12, 13, 14 (DB4 al DB7), de la pantalla LCD corresponden al 23, 24, 25, 26 (PC0 al PC3) del microcontrolador por el mismo orden. La pantalla se marca con un óvalo de color naranja en la parte central de la figura 2.2.

Figura 2.2. Interfaz serial FT232 BL, termistor (izquierda), fotorresistencia (derecha) y *LCD-Display* 162C.

Los componentes que complementa el módulo µECU se pueden ver en el Anexo IV.

Para la programación de los ejercicios que después serán montados en el módulo µECU se empleó el software Atmel Studio 7 del cual hablaremos a continuación.

2.2 Atmel Studio 7 como herramienta de programación

Atmel Studio 7 tiene una plataforma de desarrollo integrada (IDP por sus siglas en inglés) para desarrollar y depurar todas las aplicaciones de microcontroladores AVR. El IDP de Atmel Studio 7 brinda un entorno transparente y fácil de usar para escribir, construir y depurar sus aplicaciones escritas en C/C ++ o código de ensamblador. También se conecta sin problemas a los depuradores, programadores y kits de desarrollo que admiten dispositivos AVR (Technology, 2017b).

El software incluye a Atmel Gallery, una tienda de aplicaciones en línea que le permite ampliar su entorno de desarrollo con complementos desarrollados por Microchip, así como herramientas de terceros y proveedores de software integrados. Studio 7 también puede importar sin problemas sus bocetos de Arduino como proyectos de C ++ (Technology, 2017b).

- **Datos claves:**
  - Soporte para más de 500 dispositivos AVR y SAM.

- Amplia biblioteca, que incluye controladores, pilas de comunicación, más de 1600 ejemplos de proyectos con código fuente.
- Extensiones de IDP a través de Atmel Gallery.
- Ajuste y validación del rendimiento del sistema, monitoreo del consumo de energía y graficación de datos y trazas en tiempo real.
- Las características avanzadas de depuración incluyen puntos de corte de datos complejos, soporte de rastreo, perfil de código estadístico, seguimiento y monitoreo de interrupciones, rastreo de datos encuestados y seguimiento de variables en tiempo real.
- Editor integrado con asistencia visual.
- La programación y depuración en el sistema proporciona una interfaz para todos los programadores y depuradores en circuitos de Atmel.
- Crea vistas de depuración transparentes en CPU y periféricos para facilitar el desarrollo y la depuración del código.

Atmel Studio 7 diseña aplicaciones de baja potencia. Gracias a esto se puede perfilar el uso de energía de la aplicación como parte de una sesión de depuración estándar. El muestreo del contador del programa durante las mediciones de potencia permite correlacionar los picos de potencia con el código que los causó, para así lograr una buena corrección (Technology, 2017b).

El sistema de ayuda de Atmel Studio 7 admite el acceso en línea y fuera de línea, lo que significa que siempre se obtendrá la documentación más reciente cuando el usuario esté conectado. La sensibilidad del dispositivo y una vista de las E/S están contenidas en el editor, lo que permite buscar información específica del registro en la hoja de datos de la parte que se está utilizando sin salir del editor (Technology, 2017b).

Atmel Studio 7 es gratuito y está integrado con ASF (Marco de Software Avanzado), una gran biblioteca de código fuente gratuito. ASF fortalece Atmel Studio al proporcionar, en el mismo entorno, acceso a código listo para usar, el cual minimiza gran parte del diseño de bajo nivel requerido para los proyectos (Technology, 2017b). Además, el IDE de Atmel Studio 7 también:

- Facilita la reutilización del software existente y, al hacerlo, permite la diferenciación del diseño.
- Apoya el proceso de desarrollo de productos con fácil acceso a herramientas integradas y extensiones de software a través de Atmel Gallery.
- Reduce el tiempo de comercialización al proporcionar funciones avanzadas, un ecosistema de software extensible y una potente integración de depuración.

2.2.1 ¿Cómo trabajar en Atmel Studio 7?

Para iniciar la programación de un microcontrolador primeramente se debe instalar la plataforma Integrada de Desarrollo (IDP) de Atmel Studio 7.

Instalado el software, se pasa a crear el proyecto. Una vez abierto el software, en la parte izquierda de la figura 2.3 se ve un link llamado “New Project” en el cual se debe dar click.

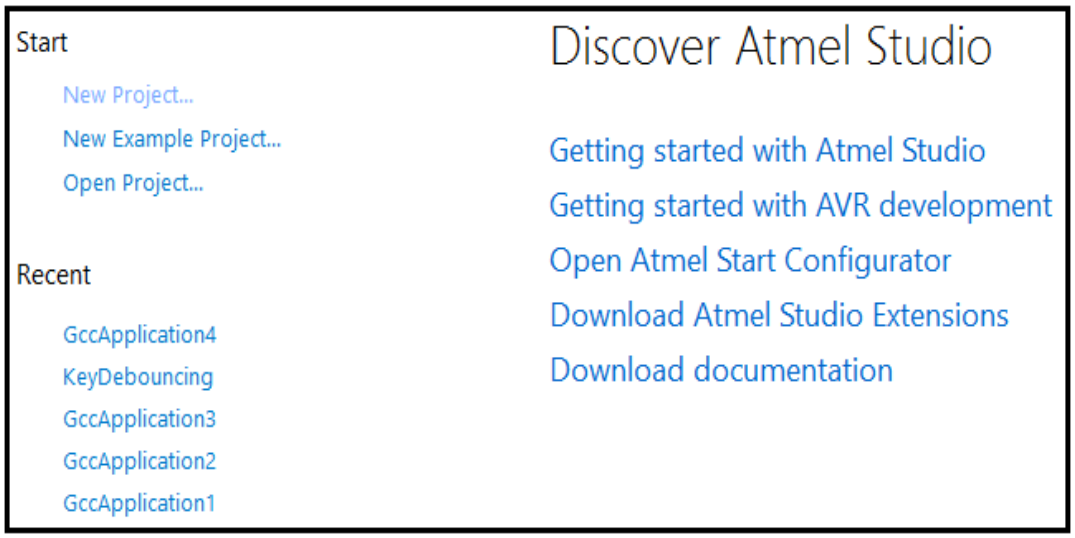


Figura 2.3. Pantalla Inicial.

Luego se debe seleccionar el tipo de proyecto y el lenguaje (C, C ++, ASM (ensamblador)). En este ejemplo se usa el lenguaje C como se ve en la figura 2.4.

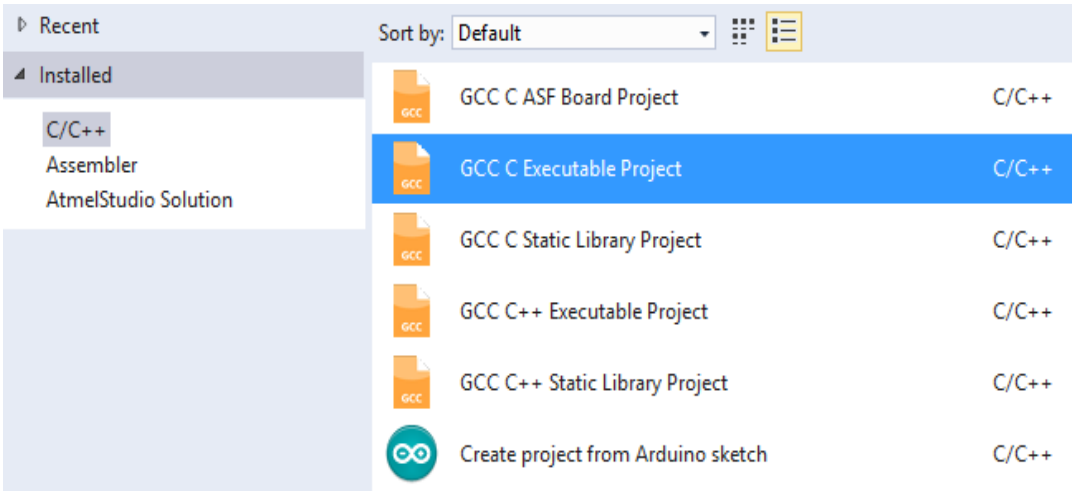


Figura 2.4. Selección del lenguaje.

Una vez seleccionado el lenguaje, se debe dar un nombre al proyecto, en este caso se llama "Primer-Proyecto" (figura 2.5). El software creará una carpeta por defecto con el nombre del proyecto en la ruta "C:\users\user\Documents\Atmel Studio\7.0". Aunque se puede crear una nueva localización si se desea.

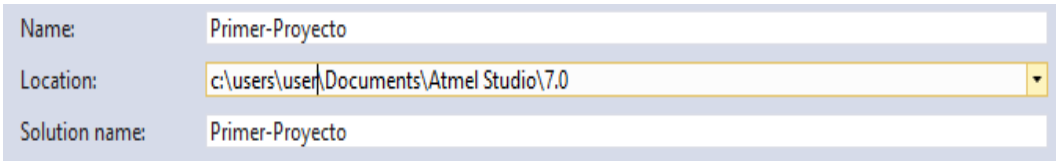


Figura 2.5. Nombre del proyecto.

Ahora se debe seleccionar el dispositivo que desea usar, en este caso se selecciona el ATmega88PA (figura 2.6).

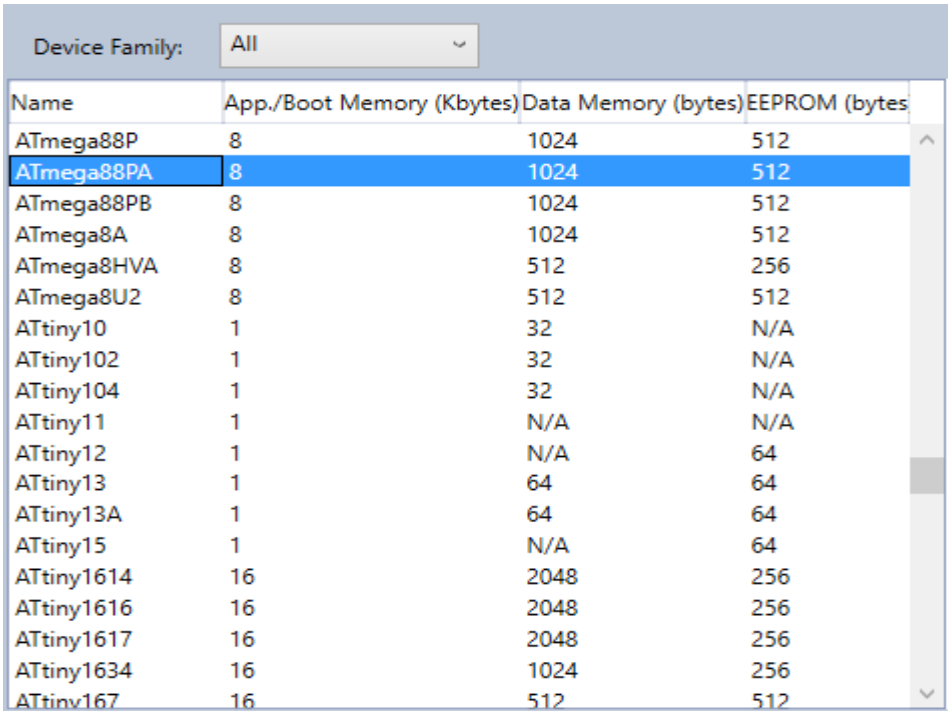


Figura 2.6. Selección del microcontrolador.

A la derecha de la misma ventana de la figura 2.6 se ve una lista de las de las herramientas de depuración y programación que se pueden usar en el microcontrolador seleccionado (figura 2.7).

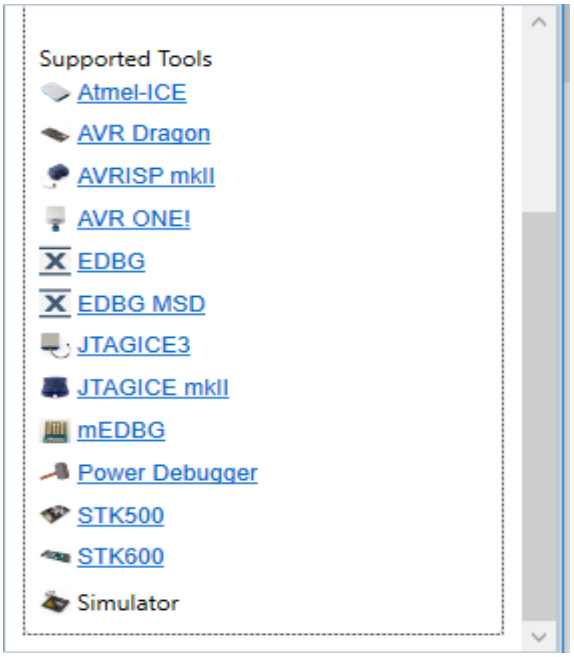


Figura 2.7. Herramientas pertenecientes al microcontrolador seleccionado.

También aparece un enlace a la hoja de datos del dispositivo (figura 2.8).

Figura 2.8. Hoja de datos del microcontrolador.

Luego de dar click en el botón ok se aprecia el archivo main.c tal como muestra la figura 2.9. Aquí es donde se debe escribir el programa.

```
/*
 * Primer-Proyecto.c
 *
 * Created: 03/10/2020 11:19:46
 * Author : Nombre
 */

#include <avr/io.h>


int main(void)
{
    /* Replace with your application code */
    while (1)
    {
    }
}
```

Figura 2.9. main.c del programa.

Después de realizar el programa deseado, se pasa a compilar para chequear que no hay errores y está bien programado mediante los botones que se señalan en un círculo rojo en la figura 2.10.

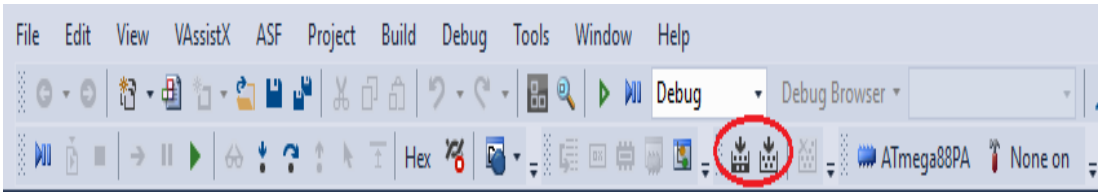


Figura 2.10. Botones de compilación.

Finalmente, se tendrá como salida el archivo hexadecimal para poder programar el microcontrolador. Se muestra en la figura 2.11 en un círculo rojo con la extensión (.hex).

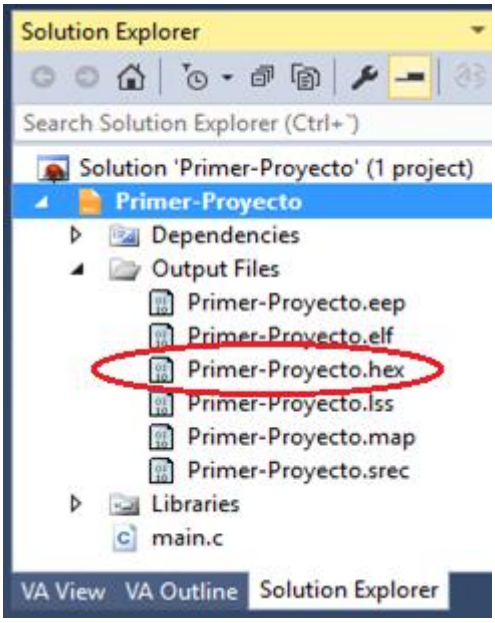


Figura 2.11. Salida en hexadecimal del proyecto.

2.3 Programación del microcontrolador ATmega88PA.

Una vez ya creado y compilado el código se carga en el microcontrolador del módulo µECU. Cuando se conecta el módulo a al equipo de cómputo, para su conexión, en el software Atmel Studio 7 se debe dar click en “Tools/Add target…” y saldrá una ventana (figura 2.12)

- “Select toool” seleccionar STK500.
- “Select USB Serial Port” seleccionar el puerto donde está conectado el módulo µECU COM3 (el puerto COM puede variar según en la computadora que se esté usando)
- Click en “Apply”

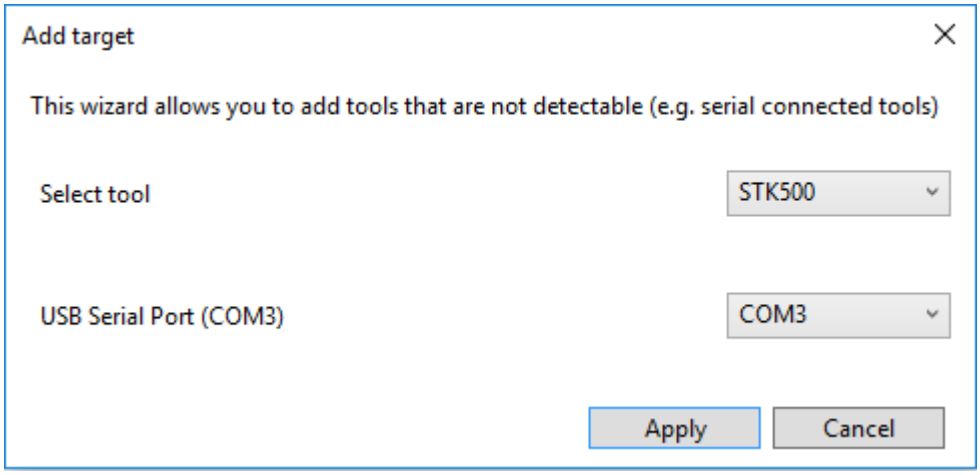


Figura 2.12. Add target.

Después se debe ir a "Tool nuevamente y seleccionar Device Programming (Ctrl+Shift+P)" donde se despliega una nueva ventana (figura 2.13), se debe configurar según el puerto COM, el microcontrolador y la interfaz.

- "Tool" STK500 COM3
- "Device" ATmega88PA
- "Interface" ISP

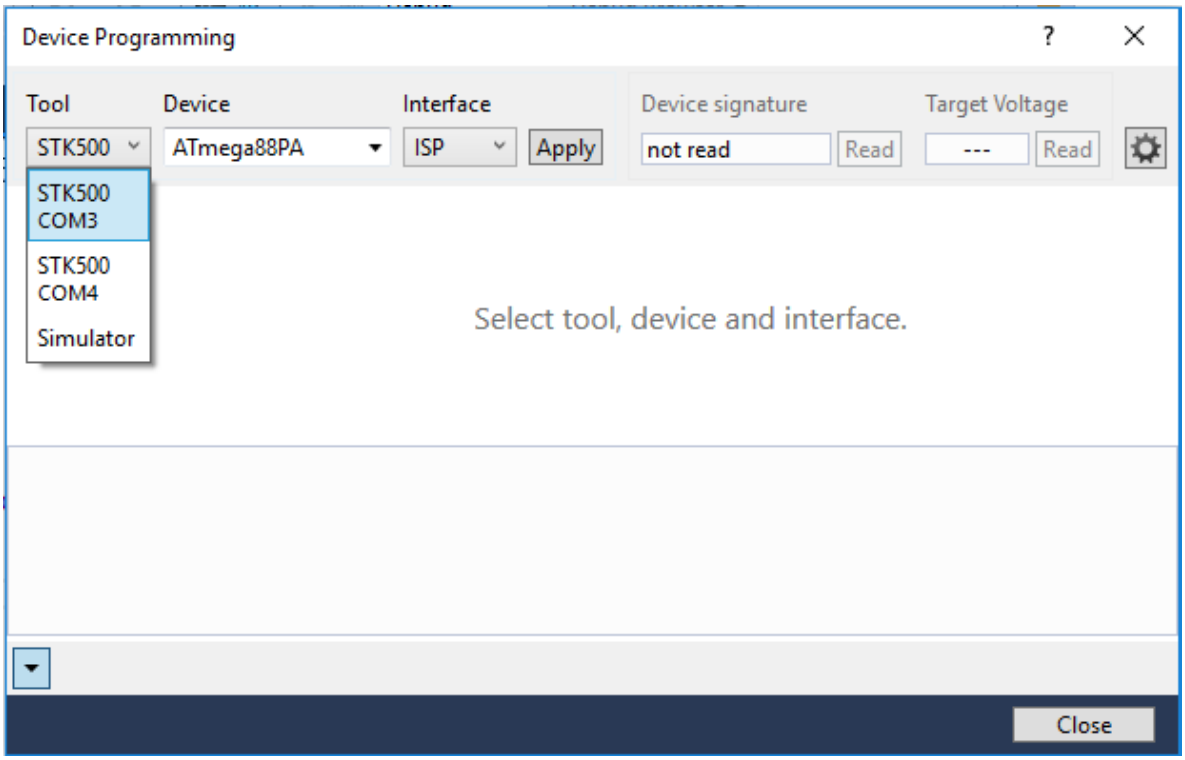


Figura 2.13. Programación del dispositivo.

Luego como se observa en la figura 2.14 al dar click en “Aplly” y seguidamente click en “Read” se reconoce el dispositivo. El microcontrolador debe estar energizado en 5V, en caso contrario no se reconocerá.

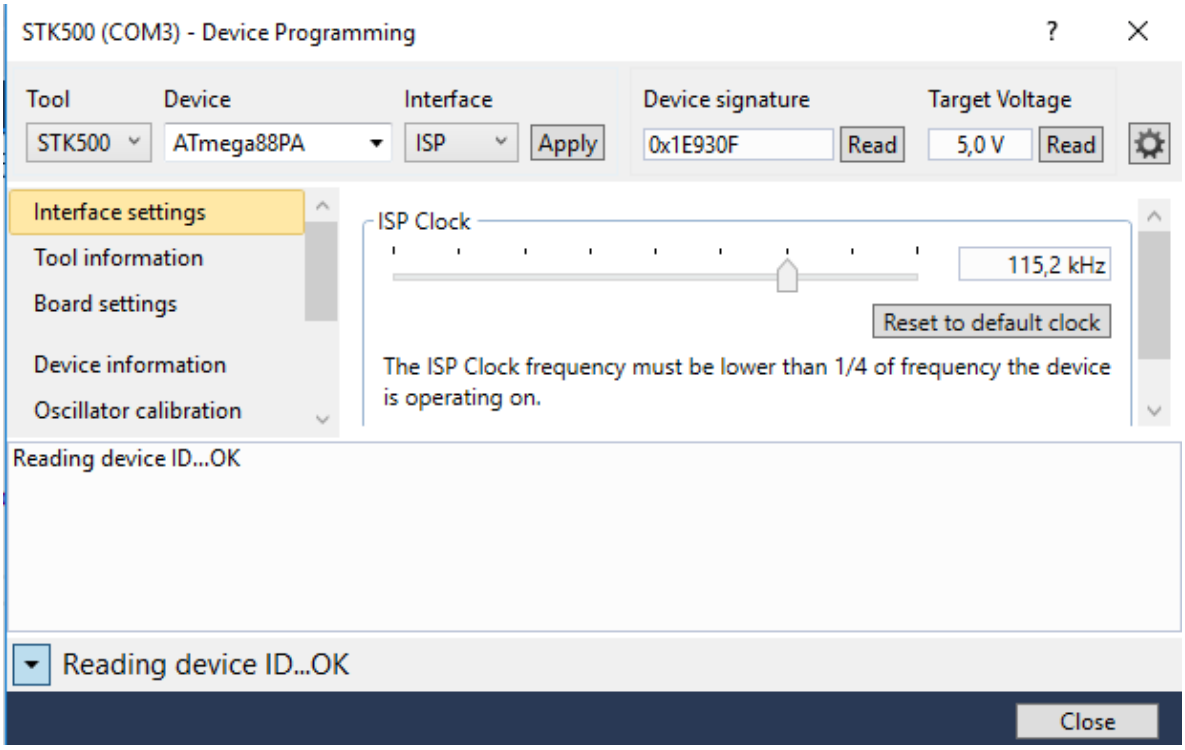


Figura 2.14. Reconocimiento del microcontrolador.

Para cargar el código al ATmega88PA dar click en “Memories”, se busca el archivo (.hex) que se generó al compilar el código, esto se hace navegando hasta la carpeta donde el usuario ha deseado guardar la compilación del programa, acto seguido se abre el archivo y se debe dar click en “Program” y asi finaliza el proceso de cargar el código en el microcontrolador (figura 2.15).

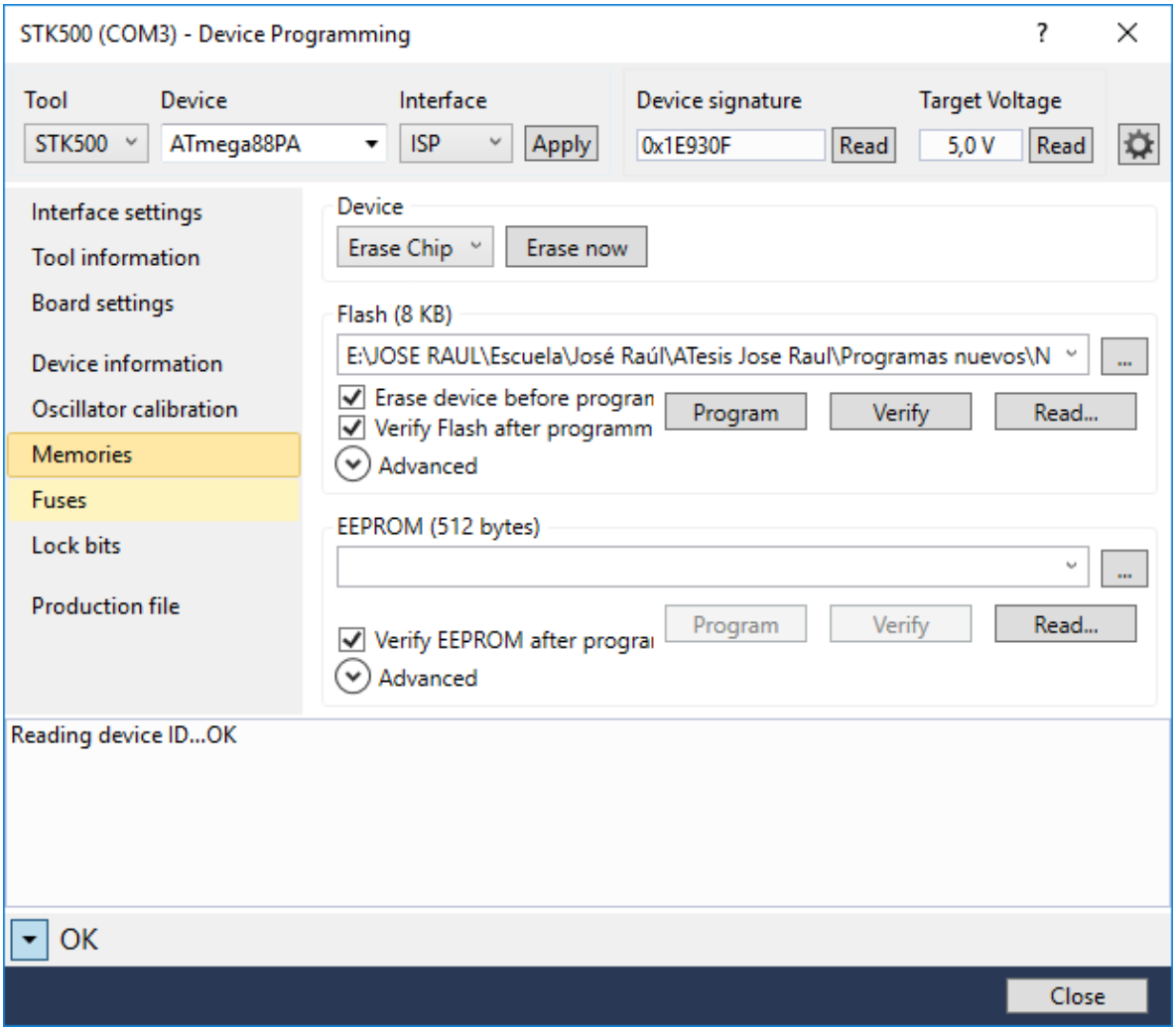


Figura 2.15. Cargar código en el microcontrolador ATmega88PA.

Conociendo los principales componentes del hardware, las características del software, cómo crear un proyecto en el mismo y como cargar los códigos en el módulo µECU se da lugar a la estructura de las guías que conformaran las prácticas de laboratorio en el presente trabajo.

2.4 Estructura de las Guías de las Prácticas de Laboratorio

El desarrollo efectivo de las prácticas de laboratorio resulta de gran importancia para el aprendizaje en varias asignaturas de la carrera, ya que estas posibilitan que el estudiante consolide los conocimientos adquiridos en el aula. Las guías para cada práctica se encuentran divididas en cuatro partes fundamentales según: objetivos, tarea preliminar, técnica operatoria y conclusiones, cuyas funciones y características se muestran a continuación:

- **Objetivos:** Se definen los objetivos de la práctica de laboratorio.
- **Tarea Preliminar:** Tiene como objetivo que el estudiante se familiarice con el tema correspondiente a la práctica de laboratorio que va a realizar. Para lograr esto, al alumno se le deben indicar una serie de tareas, como:
  - Solución de ejercicios correspondientes a la práctica de laboratorio.
- **Técnica Operatoria:** Es la parte más importante dentro de la práctica de laboratorio, pues en ésta se describe toda la operatoria que debe realizar el estudiante durante el desarrollo de la actividad práctica, se indican los análisis que debe utilizar y se culmina la programación de los programas con el montaje en el módulo µECU.
- **Conclusiones:** En esta parte se le indica al estudiante responder una serie de preguntas, lo cual le permite concluir, respecto al funcionamiento, montaje y características propias de cada uno de los ejercicios tratados en la actividad práctica.

2.5 Propuestas de prácticas a desarrollar

Con este trabajo se pretende mediante los ejercicios siguientes, los cuales son el núcleo de estas prácticas de laboratorio, cuatro en su totalidad, desarrollar el conocimiento y el manejo de habilidades en las materias que se consideran las más esenciales para empezar a adentrarse en esta tecnología.

2.5.1 Indicaciones generales de las prácticas de laboratorio

Un objetivo que se hace común en las actividades prácticas que se tratan en este trabajo es fomentar la reutilización del código por parte de los estudiantes, por lo que se le indica segmentar el código en módulos correspondientes a la funcionalidad del dispositivo. Otro elemento que coincide en estas prácticas de laboratorio es que los estudiantes deben ir con una preparación previa a la actividad. Esta preparación se logra con los contenidos impartidos en conferencias y clases prácticas. También cada proyecto se debe crear en su propia carpeta.

2.5.2 Práctica de laboratorio 1: “Manejo de botones y led del módulo µECU”

- Objetivo

Programar y emplear entradas y salidas digitales del microcontrolador ATmega88PA.

- Técnica Operatoria

Crear un programa C para que el microcontrolador haga que el led rojo se encienda sólo mientras se presiona el botón S4.

En la figura 2.16 se muestra el segmento de código que realiza la configuración del pin correspondiente al led rojo del módulo; uno de los que son accesibles del microcontrolador. Se configura el pin 2 del puerto B correspondiente al led rojo como salida.

```
void LEDInit()
 {
    DDRB |= (1<<DDB2);
 }
```

Figura 2.16. Configuración de los puertos de entrada y salida correspondientes a los leds.

En la figura 2.17 se muestra la inicialización como entrada digital de los pines del microcontrolador correspondientes a las botones de la placa µECU.

```
char keyInit()
 {
    DDRD &= ~(1<<DDD2) & ~(1<<DDD3) & ~(1<<DDD4) & ~(1<<DDD5);
 }
```

Figura 2.17. Inicialización de los pines que tributan a los botones.

En la figura 2.18 se muestra la implementación de la función para encuestar el estado de uno de los botones.

```
char keyPressed(char key)
 {  if ((key==1) && !(PIND & (1<<PD5))) return 1;
    else if ((key==2) && !(PIND & (1<<PD4))) return 1;
    else if ((key==3) && !(PIND & (1<<PD3))) return 1;
    else if ((key==4) && !(PIND & (1<<PD2))) return 1;
    else return 0;
 }
```

Figura 2.18. Función para encuestar el estado de un botón.

- Conclusiones

Se puede llegar a conclusiones dando respuestas a estas dos preguntas:

¿Cómo manejar entradas y salidas digitales?

¿Qué utilidad puede tener?

En el Anexo V se muestra en su totalidad el código correspondiente a la solución del ejercicio de esta práctica.

2.5.3 Práctica de laboratorio 2: "Configuración del temporizador del microcontrolador ATmega88PA"

- Objetivo

Configurar y emplear el módulo temporizador 1 del microcontrolador ATmega88PA.

- Técnica Operatoria

En esta práctica se reutiliza el código de la práctica anterior en lo relativo al empleo de las salidas digitales mediante leds.

Crear un proyecto C para el módulo µECU donde el temporizador 1 será inicializado para activar cada 10 milisegundos (ms) una interrupción del temporizador. Completa el programa para que el LED2 parpadee cada 1 s.

En la figura 2.19 se muestra la función en la cual se configura el temporizador 1. En un primer paso se configura para utilizar un multiplicador de ocho y como fuente del contador el reloj interno. En un segundo paso se configura el modo CTC (empieza a contar de cero y cuando llega al valor máximo (escrito en el registro OCR1A) se reinicia). Después se debe establecer el valor máximo del temporizador escribiendo en el registro OCR1A. Luego se habilita las interrupciones del temporizador. Por último, se habilita todas las interrupciones del dispositivo.

```
void Timer1_Init (void)
{
    TCCR1B = 1<<CS11;
    TCCR1B |= 1<<WGM12;
    OCR1A = 23039;
    TIMSK1 |= 1<<OCIE1A;
    sei ();
}
```

Figura 2.19. Ejemplo de función de la configuración del temporizador 1.

Para utilizar el temporizador se recomienda utilizar interrupciones en este caso se les da la implementación de la interrupción del temporizador a los estudiantes como se puede apreciar en la figura 2.20.

```
ISR (TIMER1_COMPA_vect)
{ucTimer1_10msFlag =1;
    ucTimer1_1sCnt++;
    if(ucTimer1_1sCnt ==100)
    {ucTimer1_1sFlag=1;
        ucTimer1_1sCnt =0;}
}
```

Figura 2.20. Implementación de la interrupción del temporizador.

- Conclusiones

Se permite llegar a conclusiones explicando las preguntas siguientes:

¿Qué registros empleamos en la configuración del temporizador 1?

¿Qué funcionalidad del temporizador se configuró en cada registro?

El código integro de la solución de este ejercicio correspondiente a esta práctica se encuentra en el Anexo VI.

2.5.4 Práctica de laboratorio 3: “Identificación de teclas”

En esta práctica se reutiliza el código de prácticas anteriores en lo relativo al empleo de las entradas y salidas digitales mediante botones y leds y también reutilizamos la configuración del temporizador.

- Objetivos

Emplear interrupciones.

Emplear la funcionalidad “toggle” (alternar) del microcontrolador.

- Técnica Operatoria

Crear un programa en C que cambie el estado del led verde de la placa cada vez que el usuario presione la tecla S1. El estado de la tecla debe ser encuestado aproximadamente cada 10 ms en un ciclo para evitar los ruidos por el contacto. El estado del tiempo transcurrido debe medirse con el temporizador 1 empleando una interrupción que se habilite cada 10 ms.

Para utilizar la interrupción del temporizador se deben habilitar tanto la bandera global de interrupciones como la bandera de interrupciones del temporizador, ver figura 2.21.

```
TIMSK1 |= 1<<OCIE1A;

     sei ();
```

Figura 2.21. Pasos para habilitar interrupciones.

Usando el lenguaje C la manera de utilizar la interrupción donde se emplea la macro ISR para registrar un segmento o rutina de código como interrupción, el argumento identifica la interrupción en la cual se ejecuta la rutina (figura 2.22).

```
if (Timer1_get_10msState()==TIMER_TRIGGERED)
       {
          ucKey= key_get_state();
          switch (ucKey)
          {
            case S1_PRESSED:
            LEDToggle(GREEN);
            break;
          }
       }
```

Figura 2.22. Empleo de una rutina de interrupción.

Se muestra un segmento de código en la figura 2.23 el cual encuesta el estado de una bandera que indica el transcurso de un periodo de al menos 10 ms, esta bandera se habilita en la interrupción del temporizador. Transcurrido este período se verifica el cambio de estado del botón S1. El periodo de tiempo es necesario para evitar ruidos.

```
ISR (TIMER1_COMPA_vect)
{
    ucTimer1_10msFlag =1;
    ucTimer1_1sCnt++;
    if(ucTimer1_1sCnt ==100)
    {
        ucTimer1_1sFlag=1;
        ucTimer1_1sCnt =0;
    }
}
```

Figura 2.23. Segmento de código para encuestar el estado de una bandera activada por interrupción.

En la función “LEDToggle” se hace empleo de la funcionalidad alternar del microcontrolador. En la figura 2.24 se muestra el empleo del registro PIND para usar esta funcionalidad en el pin 6 (PD6) del puerto D.

```
PIND |= (1<<PD6);
```

Figura 2.24. Código de ejemplo donde se empleó la funcionalidad alternar toggle (alternar).

- Conclusiones

Se puede concluir esta práctica con las siguientes preguntas:

¿Cómo se implementa la interrupción correspondiente al temporizador 1?

¿Mediante qué registro se emplea la funcionalidad (alternar)?

La solución del código del ejercicio en su totalidad se encuentra en el Anexo VII.

2.5.5 Práctica de laboratorio 4: “Uso de pantalla LCD”

- Objetivo

Emplear el *LCD-Display* 162C de 16x2 para la visualización de texto.

- Técnica Operatoria

Ejercicio 1

Para el módulo µECU cree un proyecto en C que muestre el siguiente texto:

M	**i**	**c**	**r**	**o**	**c**	**o**	**n**	**t**	**r**	**o**	**l**	**l**	**e**	**r**
		A	**S**	**E**	**3**		**W**	**S**	**1**	**6**	**/**	**1**	**7**	

Para este ejercicio se cuenta con una implementación de alto nivel de la funcionalidad del *display* que se encuentra implementada en los archivos “display_funktionen.h” y “display_funktionen.c” los cuales se encuentran íntegramente en el Anexo VIII.

En la figura 2.25 se muestra como se incluyen las funciones del *display* mediante #include "display_funktionen.h". Se inicializa el hardware de la pantalla LCD mediante la función

```
#include "display_funktionen.h"

int main(void)
{
    Display_Clear();
    Display_SetCursor(0,0);
    Display_Print((unsigned char *)"Microcontroller",15);
    Display_SetCursor(1,2);
    Display_Print((unsigned char *)"ASE3 WS16/17",12);
    while (1)
    {
    }
}
```

Figura 2.25. main.c del primer ejercicio.

Display_Init. Después se ve cómo se puede poner el cursor en una posición de la matriz de 2x16 mediante la función Diplay_SetCursor. Display_Print es la función para escribir, se le pasan 2 parámetros, primero los caracteres y luego la cantidad de caracteres que se desea mostrar. Se observa cómo se puede escribir en otra posición y fila diferente. Se ve la utilización del while (1) para que el microcontrolador no se reinicie constantemente.

Ejercicio 2

Pulsando la tecla S1, su nombre aparecerá en la segunda fila de la pantalla. Pulsando la tecla S2, el texto original aparecerá en la segunda línea.

El ejercicio 2 es una integración de contenidos (figura 2.26). Ya que los includes que se utilizan son las cabeceras para usar teclado, temporizadores y teclas los cuales se discutieron en prácticas anteriores. Se crearon dos funciones nuevas en #include “Pantallas.h” donde se muestra en la pantalla cierta información deseada. La función pantallaPrincipal(); está en el archivo “Pantallas.c”, es lo que muestra la primera pantalla y es lo que se muestra por defecto al iniciar el dispositivo. En el while (1) es donde se realiza el proceso principal. if (Timer1_get_10msState()==TIMER_TRIGGERED) se ve en la práctica de teclas y leds, lo que ahora se trabaja con teclas y *display*, se utiliza para que no haya ruido. Después se llama la función “key_get_state” y se guarda el resultado en ucKey, la función está en “keyDeb.c”. switch (ucKey) verifica lo que hay en “ucKey”, o sea lo que pasa con las teclas. Si lo que pasa es que se presiona S1 “case S1_PRESSED:” se muestra la pantalla principal, llamando

a la función que se crea nueva para este ejercicio “pantallaPrincipal();”. Pasa algo muy parecido a lo de la tecla S1, pero cuando se preciona la tecla S2 “case S2_PRESSED:”, y se llama a la función “pantallaNombre();”

```
#include <avr/io.h>
#include "display_funktionen.h"
#include "Timer1.h"
#include "keyDeb.h"
#include "Pantallas.h"

int main(void)
{
    unsigned char ucKey = KEYS_NOT_CHANGED;
    Display_Init();
    keyInit();
    Timer1_Init();
    pantallaPrincipal();

    while (1)
   {
       if (Timer1_get_10msState()==TIMER_TRIGGERED)
       {
          ucKey= key_get_state();
          switch (ucKey)
          {
            case S1_PRESSED:
            pantallaPrincipal();
            break;
            case S2_PRESSED:
            pantallaNombre();
            break;
          }
       }
   }

}
```

Figura 2.26. main.c del segundo ejercicio.

En la figura 2.27. se explica las dos funciones nuevas de las pantallas, como se aprecia se ponen en un archivo separado para facilitar la comprensión del código en el programa principal y la reutilización. void pantallaPrincipal() es la implementación de la función que muestra la pantalla principal. La función Display_Clear(); limpia la pantalla para garantizar que salga todo limpio. void pantallaNombre() hace lo mismo que la función anterior, lo que se separa en funciones el código para que se entienda mejor el programa.

```
void pantallaPrincipal()
{
    Display_Clear();
    Display_SetCursor(0,0);
    Display_Print((unsigned char
*)"Microcontroller",15);
    Display_SetCursor(1,2);
    Display_Print((unsigned char *)"ASE3 WS16/17",12);
}
void pantallaNombre()
{
        Display_Clear();
        Display_SetCursor(1,0);
        Display_Print((unsigned char *)"JOSE RAUL",9);
}
```

Figura 2.27. Funciones pantallaPrincipal() y pantallaNombre().

- Conclusiones

Para arribar a conclusiones se podría preguntar:

¿Es importante la visualización de datos?

¿Cómo emplear la pantalla LCD en este caso?

¿Por qué la segmentación del código facilita la reutilización y la velocidad del trabajo?

El código integro de los dos ejercicios se encuentra en el Anexo VIII.

2.6 Conclusiones parciales del capítulo

El software Atmel Studio 7 puede mejorar el manejo de habilidades en los estudiantes principalmente porque brinda un entorno transparente y fácil de usar para escribir, construir y depurar sus aplicaciones escritas en C/C ++ o código de ensamblador.

Con la proposición de las cuatro prácticas de laboratorio se permite la adquisición de conocimientos en lo relativo a entradas y salidas, temporizadores, interrupciones y el trabajo con la pantalla LCD.

CAPÍTULO 3. RESULTADOS Y DISCUSIÓN DE LAS PRÁCTICAS DE LABORATORIO

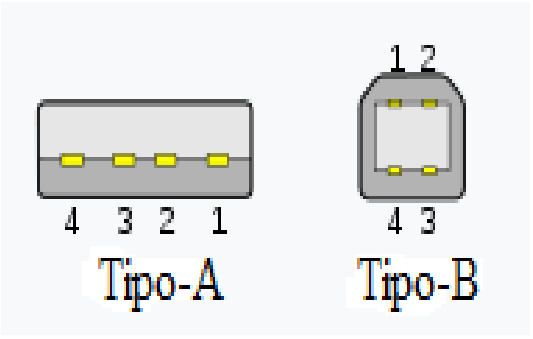
En este capítulo se exponen los resultados obtenidos satisfactoriamente de las propuestas de las prácticas realizadas. Por los resultados obtenidos se proponen asignaturas en las que se pueden emplear. Se finaliza con análisis económico y medioambiental del trabajo.

3.1 Resultados de la programación del microcontrolador ATmega88PA en el módulo µECU

Con el objetivo de evaluar la ejecución de los ejercicios propuestos, se montaron cada una de las actividades en el microcontrolador ATmega88PA del módulo µECU. Los programas fueron cargados desde una computadora Lenovo, su procesador es un core i5 de quinta generación con opciones de bajo consumo, su CPU trabaja a 2.20 GHz de frecuencia, posee una memoria RAM de 8 GB, el sistema operativo es de 64 bits y el procesador es x64, el Windows que utiliza es Windows 10 Pro. Cumple con los requisitos mínimos para trabajar con el software Atmel Studio 7.

El módulo µECU se conecta al equipo de cómputo por un cable USB 2.0 tipo A macho a tipo B macho (figura 3.1.a). En la figura 3.1.b se muestra el patillaje del cable.


a



b

Figura 3.1.(a, b). a cable USB, b patillaje.

Cuando el módulo se conecta al equipo de cómputo por el cable antes mencionado se enciende el led4 (verde) que se aprecia en la figura 3.2.a. Si el módulo no se alimenta el led4 no se enciende (figura 3.2.b).


a


b

Figura 3.2.(a, b). led4 (verde) en (a) encendido, led4 (verde) en (b) apagado.

Cuando se conecta el módulo al equipo de cómputo, y mediante el software Atmel Studio 7 el microcontrolador ATmega88PA se empieza a programar el led3 (amarillo) se enciende (figura 3.3.a) y se apaga una vez culminada la programación como se muestra en la figura 3.2.b.


a


b

Figura 3.3. (a, b). led3 (amarillo) en (b) no está siendo programado (apagado), en (a) está siendo programado (encendido).

- Práctica 1: “Manejo de botones y led del módulo µECU”

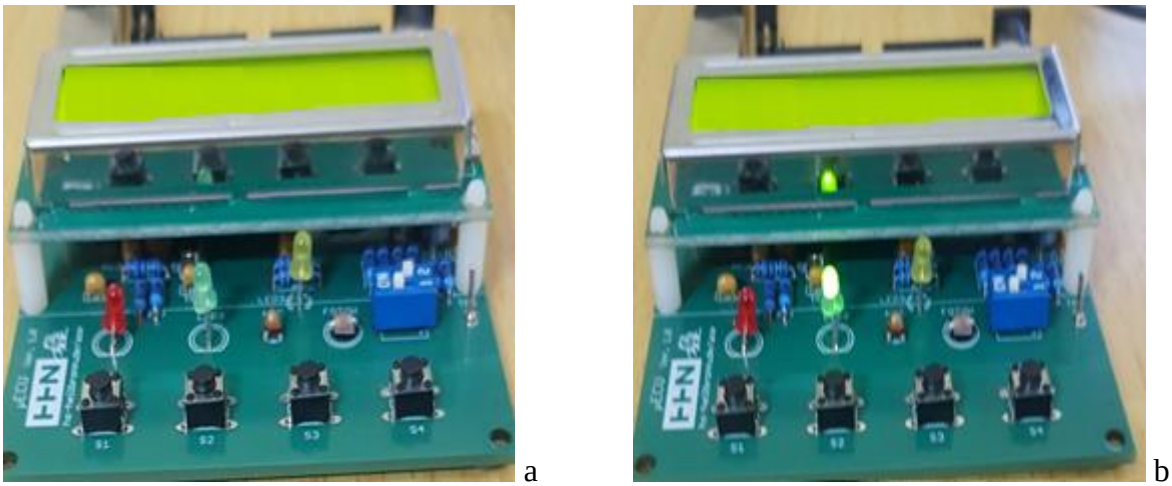
Como resultado de la primera práctica de laboratorio se muestra en la figura 3.4 que el led1 (rojo) se enciende al presionar el botón de tacto S1, al dejar de presionar el botón de tacto S1 el led1 se apaga.

Figura 3.4 led1 (rojo) se enciende al presionar el contacto S1.

- Práctica 2: “Configuración del temporizador del microcontrolador ATmega88PA”

Se aprecia en la figura 3.5.(a, b), que al programar el microcontrolador ATmega88PA del módulo con el ejercicio realizado en la práctica de laboratorio número dos, el led2 (verde) se enciende y se apaga parpadeando continuamente cada un segundo. En la figura 3.5.a el led2

(verde) está apagado, cuando pasa un segundo se enciende figura 3.5.b, después vuelve a apagarse y repite la acción cíclicamente.

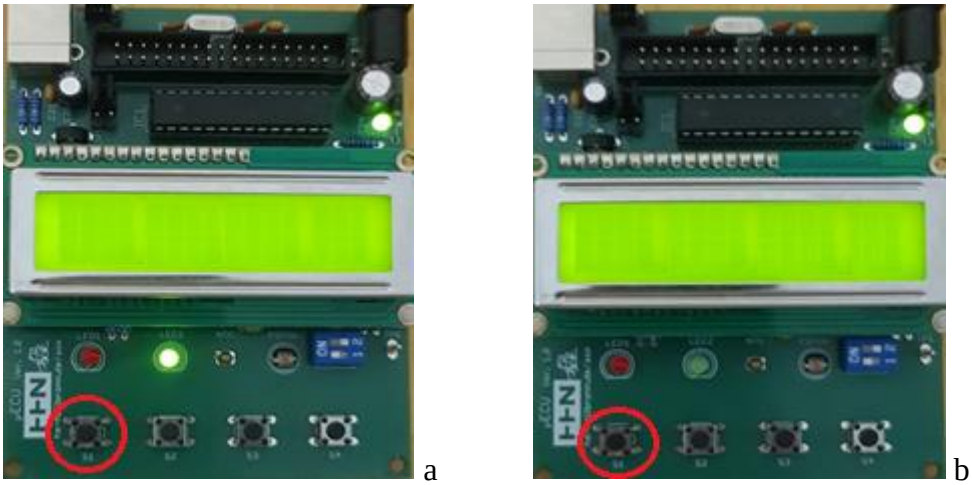


a b

Figura 3.5.(a,b). a led2 (verde) apagado, x.b led2 (verde) encendido.

- Práctica 3: “Identificación de teclas”

Los resultados obtenidos en la tercera práctica de laboratorio se observan en la figura 3.6.(a, b). Al presionar el botón de tacto S1 enmarcado en un óvalo de color rojo el led2 (verde) se enciende figura (3.6.a). En la figura 3.6.b al presionar el mismo botón de tacto S1 el led2 (verde) se apaga.



a b

Figura 3.6.(a, b). a, led2 (verde) encendido cuando se presiona S1. b, led2 (verde) apagado cuando se vuelve a presionar S1.

- Práctica 4: “Uso de la pantalla LCD”

Se propusieron dos ejercicios. En la figura 3.7 se aprecia la solución del primer ejercicio al mostrar por pantalla el texto “Microcontroller” en la primera fila y “ASE3 WS16/17” en la segunda fila.

Figura 3.7. Muestra por pantalla el texto deseado.

En el segundo ejercicio se muestra por pantalla el nombre del estudiante como se aprecia en la figura 3.8 al tocar tocar el botón de tacto S2 el cual se ve enmarcado en un ovalo de color rojo en la parte inferior de dicha figura.

Figura 3.8. Muestra por pantalla el nombre deseado al tocar el boton S1.

Al tocar el botón S1 la pantallla vuelve a mostrar el texto del primer ejrcicio (figura 3.9)

Figura 3.9. Vuelve a mostrar el texto principal al tocar el botón S1

La solución fue menos compleja gracias a la reutilización de los códigos de los programas de los ejercicios anteriores, por tanto se considera un ejercicio integrador.

3.2 Posible empleo en las asignaturas del plan de estudios

Según el plan de estudio “E”, el Ingeniero en Automática es un profesional de perfil amplio. Esto conlleva a que puede ocupar puestos de trabajo como especialista de automatización. Ya que el mismo posee una formación integral y una fuerte base en ciencias básicas, fundamentalmente en las matemáticas, así como en lo relativo a los circuitos eléctricos, la electrónica, las mediciones y en las técnicas de computación (tanto desde el punto de vista del hardware como del software de bajo y alto nivel).

Un posible empleo del módulo µECU es en proyectos que cierran asignaturas que tengan como objeto de estudio el empleo de microcontroladores. También se puede adicionar esta tecnología en una asignatura optativa que finalice con un proyecto basado en microcontroladores de la tecnología Atmel. Especialmente debe emplearse en asignaturas pertenecientes a la disciplina de sistemas de computación, principalmente en la asignatura de Microcontroladores II.

3.2.1 Posible empleo en la asignatura Microcontroladores II.

Esta asignatura forma parte del currículo base, pertenece a el tercer año de la carrera y cuenta con 64 horas clases. Estas prácticas tienen un fuerte vínculo con la asignatura porque su

realización con estos módulos hace que sea más fácil entender los siguientes objetivos generales de dicha materia:

- Describir las diferencias y semejanzas entre un microprocesador y un microcontrolador para poder efectuar una selección correcta al resolver una aplicación dada.
- Saber la arquitectura y organización interna de un microcontrolador, su repertorio de instrucciones y modos de direccionado, así como el trabajo con los sistemas empotrados en el mismo con la finalidad de diseñar sistemas sencillos basados en estos dispositivos y de uso frecuente en la automática.
- Describir las características fundamentales de otros microcontroladores con la finalidad de poder establecer comparaciones entre los mismos.
- Desarrollar programas para ser ejecutados por el microcontrolador en lenguaje C con la finalidad de establecer comparaciones con los programas desarrollados en lenguaje ensamblador en cuanto a: tiempo de desarrollo de la aplicación, tiempo de ejecución del programa por parte del microcontrolador, espacio de memoria de programa ocupado por el programa.
- Utilizar herramientas, tanto de hardware como de software, para la puesta a punto de los sistemas desarrollados con el microcontrolador estudiado, entre las que se encuentran: programas ensambladores, programas simuladores, programadores de microcontroladores, kits de entrenamiento, etc.

3.3 Análisis económico y medioambiental

De manera general los microcontroladores y plataformas de programación que se muestran como objeto de estudio en la carrera tienen un coste inicial relativamente bajo. Pero hay diferencias en precios entre unos y otros, esto es debido al implementación que brindan cada uno de ellos.

Debido al poco desarrollo sobre esta rama de la tecnología en el país no se cuenta con muchos detalles sobre factibilidad económica en el desarrollo de un módulo µECU por lo que se realiza una estimación a través del análisis desarrollado por fuentes extranjeras en referencia a los beneficios que aportan su uso al desarrollo tecnológico.

Por ejemplo, en la carrera se usa un kit basado en 8051 el cual tiene un coste que oscila de 5 a 20 USD. Se usan arduinos nanos, uno, pro micro y mega sus precios son más elevados ya que varían desde 28 a 103 USD, pero sus aplicaciones son mayores. También se ha usado raspberry pi 3b. Su precio es más elevado comparado con el kit basado en 8051, ya que sus prestaciones son mayores y conllevan a una aplicabilidad mayor, su precio es de 35 USD. Estas tecnologías son relativamente baratas como se menciona anteriormente, pero la carrera Ingeniería Automática se ha podido dotar de módulos µECU los cuales tienen un costo de 29 USD. Los tenemos en el departamento gracias a que han sido donados por una universidad alemana. No tienen la misma diversidad de aplicaciones que otros módulos usados en la carrera, pero con respecto a su precio su nivel de aplicabilidad es elevado. El módulo cumple con los requisitos mínimos e indispensables que requieren las materias que son afines con sus prestaciones y funcionamiento.

El proyecto no tiene un impacto directo en el medio ambiente. Pero si repercute en el aprovechamiento del ser humano ya que dependiendo de su uso se pueden crear aplicaciones que ayuden o interactúen con el medio ambiente. Estas aplicaciones, que sí, tienen repercusión con el medio ambiente, pueden enfocarse en la utilización de los sensores que posee el módulo µECU, tanto el sensor de temperatura como el de luminosidad.

3.4 Conclusiones parciales

Con la implementación de las prácticas de laboratorio el estudiante puede ganar conocimientos y experiencias para desarrollar programas basados en el microcontrolador estudiado que permitan resolver aplicaciones de baja y mediana complejidad.

Mediante la realización de las prácticas propuestas, el estudiante puede adquirir habilidades en el manejo del microcontrolador ATmega88PA.

Las prácticas implementadas pueden servir como base para desarrollar otras, más complejas, que permitan profundizar en el trabajo con el microcontrolador y su interacción con elementos externos.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

1

Las prácticas de laboratorio para el posible empleo en el plan E constituyen un recurso importante en su proceso de enseñanza - aprendizaje, ya que contribuye a la adquisición por parte del estudiante de nuevas habilidades en materias que abordan la tecnología Atmel, principalmente el microcontrolador ATmega88PA.

2

Con el presente trabajo se realizaron cuatro prácticas de laboratorio para su posible empleo en el plan E, caracterizándose las mismas por la actualización de contenidos teóricos, prácticos y didácticos, así como la introducción del trabajo con el módulo µECU, el software Atmel Studio 7 y el empleo de situaciones que aproximan la práctica a la realidad.

3

Con la implementación de estas prácticas de laboratorio el estudiante puede adquirir habilidades en el manejo del microcontrolador ATmega88PA. Además la implementación de las mismas sirven como base para desarrollar otras más complejas.

Recomendaciones

Se sugiere seguir explotando las bondades que brinda el módulo µECU, para realizar más prácticas de laboratorio. Una variante pude ser con ejercicios que utilicen los sensores del módulo µECU, termistor y fotoresistencia.

REFERENCIAS BIBLIOGRÁFICAS

ATMEL 2008. ATmega88PA.

BARBERÁ, O. V., P. 1996. El Trabajo Práctico en la Enseñanza de las Ciencias. *Revista Enseñanza de las Ciencias*

BARRIENTOS ROJAS, D. J. 2017. *Desarrollo del curso introducción al diseño de sistemas embebidos, utilizando el controlador TM4C123GH6PM como actualización del laboratorio de microcontroladores.*, Universidad de San Carlos de Guatemala.

BOLANAKIS, D. 2019. Estudio de Investigación en Educación sobre Microcontroladores.

DOURADO, L. 2006. Concepções e práticas dos professores de Ciências Naturais relativas à implementação integrada do trabalho laboratorial e do trabalho de campo. . *Revista Electrónica de Enseñanza de las Ciencias.*

FRANCISCO., V. E. 2019. Los Fabricantes de Microcontroladores.

GEE, B. C., S.G. 1992. *The Origen of Practical Work in the English School Science Curriculum*.

GONZÁLEZ, E. 1994. *Las prácticas de laboratorio en la formación del profesorado de Física. Tesis doctoral.*, Universidad de Valencia, España.

GUADALUPE, L. 2006. La importancia de los laboratorios. *Revista Construcción y Tecnología.*

HODSON, D. 1994. Investigación y experiencias didácticas: Hacia un enfoque más crítico del trabajo de Laboratorio. . *Revista Enseñanza de las Ciencias.*: The Ontario Institute for Studies in Education, Toronto, Canadá.

HODSON, D. 1996. *Practical work in school science: exploring some directions for changeInt.*, Int. J. Science and Education.

LAYTON, D. 1990. Student Laboratory practice and the history and philosophy of science en the student Laboratory and the science curriculum. Editado por Elizabeth Hegarty - Hazel. Londres, Reino Unido. ed.

LOCK, R. 1988. A history of practical work in schools and universities: structures and strategies still largely unexplored. The Australian Science teachers Journal, Australia. ed.

LÓPEZ, A. M. T., O.E. 2012. Las prácticas de laboratorio en la enseñanza de las ciencias naturales. *Revista Latinoamericana de Estudios Educativos.* Universidad de Caldas, Manizales.: Colombia.

MARTÍNEZ, R. P. A. 2011. *Diseño e implementacion de un control remoto seguro ante interceptación para puerta levadiza de garaje.* Pontifica Universidad Catolica del Perú.

MEROTH, A. Microcontroller Programming and Sensor Networks. 2017.

MICROCHIP, T. 2018. ATmega48PA/88PA/168PA.

MOLERO PRIETO, X. 2016. *Un viaje a la historia de la informática.*

MOLINA TUFIÑO, E. A. 2012. *Estudio, diseño e implementación de un sistema robótico pra el ruteo de cables y cielo falso, utilizado en cableado estructurado horizontal.* Universidad Tecnológica "Israel".

ORTEGA DÍAZ, S. A. 2015. *Diseño de las Prácticas de Laboratorio de La Asignatura Mediciones Eléctricas II* Universidad Central "Marta Abreu" de Las Villas.

PARRA SÁNCHEZ, C. P. 2016. *Diseño e implementación de un oscilador de 3.6 a 4.04 y de 4.3 a 5.4 GHz, para el laboratorio de la escuela de ingienería electrónica y telecomunicaciones.* Universidad Nacional de Chimborazo.

PETRE SORA, A. M. 2018. *Sensornetzwerke in Theorie und Praxis*.

ROBAYO ALCOCER, M. L. 2014. Desarrollo de aplicaciones prácticas para microcontroladores ATMEL bajo la plataforma de entrenamiento arduino.

SANTÍN, A. M. 2011. *Versión 2 del Manual para Prácticas de Laboratorio de las asignaturas de Electrónica Analógica.*, Universidad Central “Marta Abreu” de Las Villas, Cuba.

SCOLES, G. P., H. 2012. Innovación de una práctica de laboratorio docente en la asignatura química orgánica. *III Jornada de Educación Mediada por Tecnología. Sistemas de Educación Abierta y a Distancia (SEADI).*

TAPIA MORAGA, D. A. A. D., RICHARD ANTONIO. 2012. *Diseño, desarrollo y programación de sistema GPS utilizando Módulo PMB688 y Microcontrolador ATMEGA328P para rastreo vehícular en flota de la Universidad Nacional Autónoma de Nicaragua UNANManagua.*, Universidad Nacional Autónoma de Nicaragua Recinto Universitario “Rubén Darío”.

TECHNOLOGY, M. 2017a. ASF4 API Reference Manual.

TECHNOLOGY, M. 2017b. Atmel Studio.

TORRES ALAFITA, A. 2004. Empresa y Fabricantes de microcontroladores.

TORRES, L. V., M.; ZAPATA, P.; RODRÍGUEZ, J.; COLMENARES, E.; MORENO, S.; . 2013. *Las prácticas de laboratorio en la enseñanza de la química en la educación superior.*, Universidad Autónoma de Barcelona, España.

URREA, G. N., J.A.; GARCÍA, J.I.; ALVARADO, J.P.; BARRAGÁN, G.A.; HAZBÓN, O.;. 2013. *Del aula a la realidad. la importancia de los laboratorios en la formación*

del ingeniero. . Universidad Pontificia Bolivariana - World Engineering Education Forum, Medellín, Colombia.

VEGA ROSERO, J. G. O., GIOVANNI; VALDIVIESO, CARLOS 2012. Utilizacion de microcontroladores Atmel para la realización de Hardware demostrativo de diversos tipos de interrupciones internas y externas de los microcontroladores Atmel.

VILLALOBOS, S. 2012. Familias de Microcontroladores.

ANEXOS

Anexo I Familia megaAVR

Atmega48, Atmega48A, Atmega48P, Atmega48PA, Atmega8, Atmega8515, Atmega8535, Atmega88, Atmega88A, Atmega88P, **Atmega88PA**, Atmega8A, Atmega16, Atmega162, Atmega164A, Atmega164P, Atmega164PA, Atmega165P, Atmega165PA, Atmega168, Atmega168A, Atmega168P, Atmega168PA, Atmega16A, Atmega32, Atmega324A, Atmega324P, Atmega324PA, Atmega325, Atmega3250, Atmega3250A, Atmega3250P, Atmega325A, Atmega325P, Atmega325PA, Atmega328, Atmega328P, Atmega32A, Atmega64, Atmega640, Atmega644, Atmega644A, Atmega644P, Atmega644PA, Atmega645, Atmega6450, Atmega6450A, Atmega6450P, Atmega645A, Atmega645P, Atmega64A, Atmega128, Atmega1280, Atmega1281, Atmega1284, Atmega1284P, Atmega128A, Atmega2560, Atmega2561, AT90CAN128, AT90CAN32, AT90CAN64, Atemega16M1, Atemega32M1, Atemega64M1, AT90PWM1, AT90PWM216, AT90PWM2B, AT90PWM316, AT90PWM3B, AT90PWM81, AT90PWM161, AT90USB1286, AT90USB1287, AT90USB162, AT90USB646, AT90USB647, AT90USB82, Atmega32U2, Atmega32U4, Atmega8U2, Atmega169A, Atmega169P, Atmega169PA, Atmega329, Atmega3290, Atmega3290A, Atmega3290P, Atmega329A, Atmega329P, Atmega329PA, Atmega649, Atmega6490, Atmega6490A, Atmega6490P, Atmega649A, Atmega649P.

Anexo II Familia tinyAVR

Attiny1634, Attiny4, Attiny5, Attiny9, Attiny10, Attiny13A, Attiny13, Attiny20, Attiny40, Attiny24A, Attiny24, Attiny44A, Attiny44, Attiny84A, Attiny84, Attiny25, Attiny45, Attiny85, Attiny261A, Attiny261, Attiny461A, Attiny461, Attiny861A, Attiny861, Attiny26, Attiny2313A, Attiny2313, Attiny4313, Attiny43U, Attiny28L, Attiny48, Attiny88, Attiny87, Attiny167.

Anexo III Leyenda

Tabla 1.4: Leyenda

Color/Configuración/Español	Color/Configuración/Español
Rojo: Power (Alimentación)	Azul: Digital (Digital)
Negro: Ground (Tierra)	Amarillo: Analog (Analógica)
Verde: Programing/debug(Programación/Depurar)	Naranja: Crystal/CLK(Cristal/Reloj)

Anexo IV Componentes que complementan el módulo µECU.

Tres diodos rectificadores 1N4004, un diodo 1N4148, un zócalo IC de 28 pines, un regulador de voltaje de 7805 µA, un recortador con protección de polaridad inversa de 2.5 KΩ, dos condensadores electrolíticos de 100 µF, un transistor BC337, un conector de caja de 34 pines, una barra de apilamiento de 16 pines, una barra de apilamiento de 1x3 pines, una barra de apilamiento de 2x3 pines, un tapón hueco con protección de polaridad inversa, un condensador electrolítico de 220 µF. Cuenta además con un microcontrolador ATmega8 encargado de ser el programador del microcontrolador de dicho módulo.

Anexo V Respuesta del ejercicio correspondiente a la práctica 1

- main.c

```
#include <avr/io.h>
#include "LED.h"
#include "key.h"

int main(void)
{
        LEDInit();
    while(1)
    {
                if (keyPressed(4))
                        LEDOn(RED);
                else
                        LEDOff(RED);
    }
}
```

- LED.h

```
#ifndef LED_H_
#define LED_H_
#define RED 0
#define GREEN 1

void LEDOn(char color);
void LEDOff(char color);
void LEDToggle(char color);
void LEDInit();

#endif
```

- LED.c

```
#include <avr/io.h>
#include "LED.h"

 void LEDOn(char color)
 {
        if (color == GREEN)
                PORTD |= (1<<PD6);
        else if (color == RED)
                PORTB |= (1<<PB2);
 }
 void LEDOff(char color)
```

```
{
	if (color == GREEN)
		PORTD &= ~(1<<PD6);
	else if (color == RED)
		PORTB &= ~(1<<PB2);
}
void LEDToggle(char color)
{
	if (color == GREEN)
		PIND |= (1<<PD6);
	else if (color == RED)
		PINB |= (1<<PB2);

}
void LEDInit()
{
	DDRD |= (1<<DDD6);
	DDRB |= (1<<DDD2);
}
```

- key.h

```
#ifndef KEY_H_
#define KEY_H_

char keyInit();
char keyPressed(char key);


#endif
```

- key.c

```
#include <avr/io.h>

char keyInit()
{
	DDRD &= ~(1<<DDD2) & ~(1<<DDD3) & ~(1<<DDD4) & ~(1<<DDD5);
}
char keyPressed(char key)
{
	if ((key==1) && !(PIND & (1<<PD5))) return 1;
	else if ((key==2) && !(PIND & (1<<PD4))) return 1;
	else if ((key==3) && !(PIND & (1<<PD3))) return 1;
	else if ((key==4) && !(PIND & (1<<PD2))) return 1;
	else return 0;
}
```

Anexo VI Respuesta del ejercicio correspondiente a la práctica 2

- main.c

```
#include <avr/io.h>
#include "Timer1.h"
#include "Led.h"

int main(void)
{

Timer1_Init();
LEDInit();

   while (1)
   {
            if (Timer1_get_1sState()==TIMER_TRIGGERED)
            {
                  LEDToggle(GREEN);
            }
   }
}
```

- Timer1.h

```
#include <avr/io.h>
#include <avr/interrupt.h>
#ifndef TIMER1_H_
#define TIMER1_H_

void Timer1_Init(void);

unsigned char Timer1_get_10msState (void);
unsigned char Timer1_get_1sState (void);

#define TIMER_RUNNING 0
#define TIMER_TRIGGERED 1
#endif
```

- Timer1.c

```
#include "Timer1.h"

unsigned char ucTimer1_10msFlag=0;
unsigned char ucTimer1_1sFlag =0, ucTimer1_1sCnt=0;

void Timer1_Init (void)
```

```
{
	TCCR1B = 1<<CS11;
	TCCR1B |= 1<<WGM12;
	OCR1A = 23039;
	TIMSK1 |= 1<<OCIE1A;
	sei ();
}

ISR (TIMER1_COMPA_vect)
{
	ucTimer1_10msFlag =1;
	ucTimer1_1sCnt++;

	if(ucTimer1_1sCnt ==100)
	{
		ucTimer1_1sFlag=1;
		ucTimer1_1sCnt =0;
	}
}

unsigned char Timer1_get_1sState(void)
{
	if (ucTimer1_1sFlag ==1)
	{
		ucTimer1_1sFlag = 0;
		return TIMER_TRIGGERED;
	}
	return TIMER_RUNNING;
}
```

- Led.h

```
#ifndef LED_H_
#define LED_H_
#define RED 0
#define GREEN 1

void LEDInit();
void LEDToggle(char color);

#endif
```

- Led.c

```
#include <avr/io.h>
#include "LED.h"
```

```
void LEDInit()
{
	DDRD |= (1<<DDD6);
	DDRB |= (1<<DDD2);
}
void LEDToggle(char color)
{
	if (color == GREEN)
		PIND |= (1<<PD6);
	else if (color == RED)
		PINB |= (1<<PB2);
}
```

Anexo VII Respuesta del ejercicio correspondiente a la práctica 3

- main.c

```
#include <avr/io.h>
#include "keyDeb.h"
#include "Led.h"
#include "Timer1.h"

int main(void)
{
  Timer1_Init();
  LEDInit();
  keyInit();

unsigned char ucKey;

  while (1)
  {
        if (Timer1_get_10msState()==TIMER_TRIGGERED)
        {
              ucKey= key_get_state();
              switch (ucKey)
              {
                    case S1_PRESSED:
                    LEDToggle(GREEN);
                    break;
               }
        }
  }

}
```

- Led.h (reutilización de código)

```
#ifndef LED_H_
#define LED_H_
#define RED 0
#define GREEN 1

void LEDInit();
void LEDToggle(char color);

#endif
```

- Led.c (reutilización de código)

```
#include <avr/io.h>
#include "LED.h"

void LEDInit()
{
	DDRD |= (1<<DDD6);
	DDRB |= (1<<DDD2);
}

void LEDToggle(char color)
{
	if (color == GREEN)
		PIND |= (1<<PD6);
	else if (color == RED)
		PINB |= (1<<PB2);
}
```

- keys_better.h

```
#define KEY_S1		0x20
#define KEY_S2		0x10
#define KEY_S3		0x08
#define KEY_S4		0x04
#define KEYS		0x3C

#define EVENT_VOID 0
#define EVENT_S1_PRESSED '1'
#define EVENT_S1_RELEASED '2'
#define EVENT_S2_PRESSED '3'
#define EVENT_S2_RELEASED '4'
#define EVENT_S3_PRESSED '5'
#define EVENT_S3_RELEASED '6'
#define EVENT_S4_PRESSED '7'
#define EVENT_S4_RELEASED '8'

void keys_Init(void);
unsigned char keyCheckEvent(void);
```

- keys_better.c

```
#include <avr/io.h>
#include "keys.h"

unsigned char ucKeyNew = KEYS;
unsigned char ucKeyOld;
void keys_Init(void)
{
	DDRD &= ~(1 << DDD5);
	DDRD &= ~(1 << DDD4);
	DDRD &= ~(1 << DDD3);
	DDRD &= ~(1 << DDD2);
}
unsigned char keyCheckEvent(void)
{
	ucKeyOld = ucKeyNew;
	ucKeyNew = PIND & KEYS;
	if(ucKeyNew != ucKeyOld)
	{
		if(!(ucKeyNew & KEY_S1) && (ucKeyOld & KEY_S1))
			return EVENT_S1_PRESSED;

		else if((ucKeyNew & KEY_S1) && !(ucKeyOld & KEY_S1))
			return EVENT_S1_RELEASED;

		else if(!(ucKeyNew & KEY_S2) && (ucKeyOld & KEY_S2))
			return EVENT_S2_PRESSED;

		else if((ucKeyNew & KEY_S2) && !(ucKeyOld & KEY_S2))
			return EVENT_S2_RELEASED;

		else if(!(ucKeyNew & KEY_S3) && (ucKeyOld & KEY_S3))
			return EVENT_S3_PRESSED;

		else if((ucKeyNew & KEY_S3) && !(ucKeyOld & KEY_S3))
			return EVENT_S3_RELEASED;

		else if(!(ucKeyNew & KEY_S4) && (ucKeyOld & KEY_S4))
			return EVENT_S4_PRESSED;

		else if((ucKeyNew & KEY_S4) && !(ucKeyOld & KEY_S4))
			return EVENT_S4_RELEASED;
	}
    return EVENT_VOID;
}
```

- keys_Deb.h

```
#ifndef KEYDEB_H_
#define KEYDEB_H_

#define KEYS_NOT_CHANGED 0
#define S1_PRESSED 1
#define S1_RELEASED 2

char keyInit();
char keyPressed(char key);
unsigned char key_get_state (void);

#endif
```

- keys_Deb.c

```
#include <avr/io.h>
#include "keyDeb.h"

 unsigned char S1_new= 0xFF, S1_old = 0xFF;

  char keyInit()
 {
        DDRD &= ~(1<<DDD2) & ~(1<<DDD3) & ~(1<<DDD4) & ~(1<<DDD5);
 }

 unsigned char key_get_state (void)
{
        S1_old= S1_new;
        S1_new=PIND & (1<<PD5);
        if (!(S1_new) && (S1_old))
        {
                return S1_PRESSED;
        }
        else if (S1_new && !(S1_old))
        {
                return S1_RELEASED;
        }
        return KEYS_NOT_CHANGED;
 }

 char keyPressed(char key)
{
        if ((key==1) && !(PIND & (1<<PD5))) return 1;
        else if ((key==2) && !(PIND & (1<<PD4))) return 1;
```

```
        else if ((key==3) && !(PIND & (1<<PD3))) return 1;
        else if ((key==4) && !(PIND & (1<<PD2))) return 1;
        else return 0;
}
```

Anexo VIII Respuestas de los ejercicios correspondientes a la práctica 4

- Ejercicio 1
  - main.c

```
#include <avr/io.h>
#include "display_funktionen.h"

int main(void)
{
        Display_Clear();
        Display_SetCursor(0,0);
        Display_Print((unsigned char *)"Microcontroller",15);
        Display_SetCursor(1,2);
        Display_Print((unsigned char *)"ASE3 WS16/17",12);
    while (1)
    {
    }
}
```

  - display_funktionen.h

```
#include <avr/io.h>
#include <avr/interrupt.h>

void Display_delay(unsigned long delay_time_us);

void Display_Aus(void);
void Display_An(void);
void Display_Init(void);

void Display_HardwareInit(void);

void Display_RS_Output(void);
void Display_RS_High(void);
void Display_RS_Low(void);

void Display_EN_Output(void);
void Display_EN_High(void);
void Display_EN_Low(void);

void Display_DATA_Output(void);
void Display_DATA_BitHigh(unsigned char DataBit);
void Display_DATA_BitLow(unsigned char DataBit);

void Display_Clear(void);
void Display_ReturnHome(void);
```

```
void Display_ModeEntry(unsigned char Options);
void Display_Control(unsigned char Options);
void Display_CursorOrDisplayShift(unsigned char Options);
void Display_SetMPUInterface(unsigned char Options);
void Display_SetCursor(unsigned char row, unsigned char column);
void Display_Transfer4BitData(unsigned char Data);
void Display_Write(unsigned char ASCII_of_char);
void Display_Print(unsigned char* text2print, unsigned char length);
void Display_GenerateNewChar(unsigned char address, unsigned char*
Pattern_New_Char);

#define DISPLAY_CLEAR_FUNCTION                  0x01
#define DISPLAY_CLEAR_DISPLAY_DELAY             4000

#define DISPLAY_RETURN_HOME_FUNCTION            0x02
#define DISPLAY_RETURN_HOME_DELAY               2000

#define DISPLAY_MODE_INCR_SHIFT_OFF             0x06
#define DISPLAY_MODE_INCR_SHIFT_ON              0x07
#define DISPLAY_MODE_DECR_SHIFT_OFF             0x04
#define DISPLAY_MODE_DECR_SHIFT_ON              0x05
#define DISPLAY_MODE_DELAY                      50

#define DISPLAY_OFF                             0x08
#define DISPLAY_ON_CURSOR_OFF                   0x0C
#define DISPLAY_ON_CURSOR_ON_BLINK_OFF          0x0E
#define DISPLAY_ON_CURSOR_ON_BLINK_ON           0x0F
#define DISPLAY_CONTROL_DELAY                   50

#define DISPLAY_SHIFT_CURSOR_RECHTS             0x14
#define DISPLAY_SHIFT_CURSOR_LINKS              0x10
#define DISPLAY_SHIFT_DISPLAY_RECHTS            0x1C
#define DISPLAY_SHIFT_DISPLAY_LINKS             0x18
#define DISPLAY_CURSOR_OR_DISPLAY_SHIFT_DELAY   50

#define DISPLAY_MPU_4BIT_2_LINES_5x7_DOTS       0x28
#define DISPLAY_SET_MPU_INTERFACE_DELAY         50

#define DISPLAY_FUNKTION_SET_DDRAM_ADRESSE      0x80
#define DISPLAY_FUNKTION_SET_CGRAM_ADRESSE      0x40
#define DISPLAY_SET_RAM_ADRESSE_DELAY           50

#define F_CPU 18432000UL

#define DISPLAY_SW_DDR_REG          DDRD
#define DISPLAY_SW_PORT_REG         PORTD
#define DISPLAY_SW_BIT              PD7
```

```
#define DISPLAY_RS_DDR_REG          DDRB
#define DISPLAY_RS_PORT_REG         PORTB
#define DISPLAY_RS_BIT              PB0

#define DISPLAY_EN_DDR_REG          DDRB
#define DISPLAY_EN_PORT_REG         PORTB
#define DISPLAY_EN_BIT              PB1

#define DISPLAY_DATA_DDR_REG        DDRC
#define DISPLAY_DATA_PORT_REG       PORTC
#define DISPLAY_DATA_DB7_BIT        PC3
#define DISPLAY_DATA_DB6_BIT        PC2
#define DISPLAY_DATA_DB5_BIT        PC1
#define DISPLAY_DATA_DB4_BIT        PC0

#define DATAPORTMASKE (~((1 << DISPLAY_DATA_DB7_BIT) | (1 << DISPLAY_DATA_DB6_BIT) | (1 << DISPLAY_DATA_DB5_BIT) | (1 << DISPLAY_DATA_DB4_BIT)))
```

- display_funktionen.c

```
#include "display_funktionen.h"


#define _1US_DELAY (20000000/F_CPU)

unsigned char ucDataBit[8] = {DISPLAY_DATA_DB7_BIT, DISPLAY_DATA_DB6_BIT, DISPLAY_DATA_DB5_BIT, DISPLAY_DATA_DB4_BIT, DISPLAY_DATA_DB7_BIT, DISPLAY_DATA_DB6_BIT, DISPLAY_DATA_DB5_BIT, DISPLAY_DATA_DB4_BIT};

unsigned char ucRAM_OffsetAdresse[4] = {0x00, 0x40, 0x10, 0x50};

void Display_HardwareInit(void)
{
	Display_DATA_Output();
	Display_RS_Output();
	Display_EN_Output();
}
void Display_delay(unsigned long delay_time_us)
{
	unsigned long delay_time;

	delay_time = delay_time_us / _1US_DELAY;
```

```
        for(unsigned long j = 0; j < delay_time; j++)
        {
        }
}
void Display_Aus(void)
{
        DISPLAY_SW_DDR_REG |= 1 << DISPLAY_SW_BIT;
        DISPLAY_SW_PORT_REG &= ~(1 << DISPLAY_SW_BIT);
}
void Display_An(void)
{
        DISPLAY_SW_PORT_REG |= 1 << DISPLAY_SW_BIT;
}
void Display_RS_Output(void)
{
        DISPLAY_RS_DDR_REG |= 1 << DISPLAY_RS_BIT;
}
void Display_RS_High(void)
{
        DISPLAY_RS_PORT_REG |= 1 << DISPLAY_RS_BIT;
}
void Display_RS_Low(void)
{
        DISPLAY_RS_PORT_REG &= ~(1 << DISPLAY_RS_BIT);
}
void Display_EN_Output(void)
{
        DISPLAY_EN_DDR_REG |= 1 << DISPLAY_EN_BIT;
}
void Display_EN_High(void)
{
        DISPLAY_EN_PORT_REG |= 1 << DISPLAY_EN_BIT;
}
void Display_EN_Low(void)
{
        DISPLAY_EN_PORT_REG &= ~(1 << DISPLAY_EN_BIT);
}
void Display_DATA_Output(void)
{
        DISPLAY_DATA_DDR_REG |= (1 << DISPLAY_DATA_DB7_BIT) | (1 <<
DISPLAY_DATA_DB6_BIT) |
                                (1 << DISPLAY_DATA_DB5_BIT) | (1 <<
DISPLAY_DATA_DB4_BIT);
}
void Display_DATA_BitHigh(unsigned char DataBit)
{
        DISPLAY_DATA_PORT_REG |= (1 << DataBit);
```

```
}
void Display_DATA_BitLow(unsigned char DataBit)
{
	DISPLAY_DATA_PORT_REG &= ~(1 << DataBit);
}
void Display_Clear(void)
{
	Display_HardwareInit();
	Display_RS_Low();
	Display_Transfer4BitData(DISPLAY_CLEAR_FUNCTION);
	Display_RS_High();
	Display_delay(DISPLAY_CLEAR_DISPLAY_DELAY);
void Display_ReturnHome(void)
{
	Display_HardwareInit();
	Display_RS_Low();
	Display_Transfer4BitData(DISPLAY_CLEAR_FUNCTION);
	Display_RS_High();
	Display_delay(DISPLAY_RETURN_HOME_DELAY);
}
void Display_ModeEntry(unsigned char Options)
{
	Display_HardwareInit();
	Display_RS_Low();
	Display_Transfer4BitData(Options);
	Display_RS_High();
	Display_delay(DISPLAY_MODE_DELAY);
}
void Display_Control(unsigned char Options)
{
	Display_HardwareInit();
	Display_RS_Low();
	Display_Transfer4BitData(Options);
	Display_RS_High();
	Display_delay(DISPLAY_CONTROL_DELAY);
}
void Display_CursorOrDisplayShift(unsigned char Options)
{
	Display_HardwareInit();
	Display_RS_Low();
	Display_Transfer4BitData(Options);
	Display_RS_High();
	Display_delay(DISPLAY_CURSOR_OR_DISPLAY_SHIFT_DELAY);
}
void Display_SetMPUInterface(unsigned char Options)
{
	unsigned char dummy = 0x80, i;
```

```
	Display_HardwareInit();
	Display_RS_Low();
	Display_EN_High();

	for(i = 0; i < 4; i++)
	{
		if(Options & dummy)
		{
			Display_DATA_BitHigh(ucDataBit[i]);
		}
		else
		{
			Display_DATA_BitLow(ucDataBit[i]);
		}

		dummy = dummy >> 1;
	}

	Display_EN_Low();
	Display_EN_High();
	Display_Transfer4BitData(Options);
	Display_RS_High();
	Display_delay(DISPLAY_SET_MPU_INTERFACE_DELAY);
}
void Display_SetCursor(unsigned char row, unsigned char column)
{
	unsigned char adresse = DISPLAY_FUNKTION_SET_DDRAM_ADRESSE;

	adresse += ucRAM_OffsetAdresse[row] + column;

	Display_HardwareInit();
	Display_RS_Low();
	Display_Transfer4BitData(adresse);
	Display_RS_High();
	Display_delay(DISPLAY_SET_RAM_ADRESSE_DELAY);
}
void Display_Transfer4BitData(unsigned char _8BitData)
{
	unsigned char dummy = 0x80, i;

	Display_EN_High();

	for(i = 0; i < 8; i++)
	{
		if(_8BitData & dummy)
		{
			Display_DATA_BitHigh(ucDataBit[i]);
```

```
            }
            else
            {
                Display_DATA_BitLow(ucDataBit[i]);
            }

            dummy = dummy >> 1;
            if(i == 3)
            {
                Display_EN_Low();
                Display_EN_High();
            }
        }

        Display_EN_Low();
}
void Display_Init(void)
{
        Display_Aus();
        Display_delay(50000);
        Display_An();
        Display_delay(30000);
        Display_SetMPUInterface(DISPLAY_MPU_4BIT_2_LINES_5x7_DOTS);
        Display_Clear();
        Display_ModeEntry(DISPLAY_MODE_INCR_SHIFT_OFF);
        Display_Control(DISPLAY_ON_CURSOR_ON_BLINK_OFF);
}
void Display_Write(unsigned char ASCII_of_char)
{
        Display_HardwareInit();
        Display_RS_High();
        Display_Transfer4BitData(ASCII_of_char);
        Display_RS_Low();
        Display_delay(DISPLAY_SET_RAM_ADRESSE_DELAY);
}
void Display_Print(unsigned char* text2print, unsigned char length)
{
        unsigned char i = 0;

        Display_HardwareInit();
        Display_RS_High();

        for(i = 0; i < length; i++)
        {
            Display_Write(text2print[i]);
        }
        Display_RS_Low();
```

```
}

void Display_GenerateNewChar(unsigned char address, unsigned char* Pattern_New_Char)
{
	unsigned char adresse = DISPLAY_FUNKTION_SET_CGRAM_ADRESSE;
	unsigned char i;

	adresse |= address << 3;
	Display_HardwareInit();
	Display_RS_Low();
	Display_Transfer4BitData(adresse);
	Display_RS_High();
	Display_delay(DISPLAY_SET_RAM_ADRESSE_DELAY);

	for(i = 0; i < 8; i++)
	{
		Display_Write(Pattern_New_Char[i]);
	}
	Display_RS_Low();
}
```

- Ejercicio 2
  - main.c

```
#include <avr/io.h>
#include "display_funktionen.h"
#include "Timer1.h"
#include "keyDeb.h"
#include "Pantallas.h"

int main(void)
{
	unsigned char ucKey = KEYS_NOT_CHANGED;
	// inicializacion
   Display_Init();
	keyInit();
	Timer1_Init();
	// parte 1
	pantallaPrincipal();
	//parte2
   while (1)
  {
		if (Timer1_get_10msState()==TIMER_TRIGGERED)
		{
			ucKey= key_get_state();
			switch (ucKey)
			{
```

```
                    case S1_PRESSED:
                    pantallaPrincipal();
                    break;
                    case S2_PRESSED:
                    pantallaNombre();
                    break;
                }
            }
    }

}
```

- Pantallas.h

```
#ifndef PANTALLAS_H_
#define PANTALLAS_H_

#include "display_funktionen.h"

void pantallaPrincipal();
void pantallaNombre();

#endif
```

- Pantallas.c

```
#include "Pantallas.h"

void pantallaPrincipal()
{
        Display_Clear();
        Display_SetCursor(0,0);
        Display_Print((unsigned char *)"Microcontroller",15);
        Display_SetCursor(1,2);
        Display_Print((unsigned char *)"ASE3 WS16/17",12);
}

void pantallaNombre()
{
                Display_Clear();
                Display_SetCursor(1,0);
                Display_Print((unsigned char *)"JOSE RAUL",9);
}
```

- Timer.h (reutilización de código)

```
#ifndef TIMER1_H_
```

```
#define TIMER1_H_

#include <avr/io.h>
#include <avr/interrupt.h>

void Timer1_Init(void);

unsigned char Timer1_get_10msState (void);
unsigned char Timer1_get_1sState (void);

#define TIMER_RUNNING 0
#define TIMER_TRIGGERED 1

#endif
```

- Timer.c

```
#include "Timer1.h"

unsigned char ucTimer1_10msFlag=0;
unsigned char ucTimer1_1sFlag =0, ucTimer1_1sCnt=0;

void Timer1_Init (void)
{
	TCCR1B = 1<<CS11;
	TCCR1B |= 1<<WGM12;
	OCR1A = 23039;
	TIMSK1 |= 1<<OCIE1A;
	sei ();
}

ISR (TIMER1_COMPA_vect)
{
	ucTimer1_10msFlag =1;
	ucTimer1_1sCnt++;

	if(ucTimer1_1sCnt ==100)
	{
		ucTimer1_1sFlag=1;
		ucTimer1_1sCnt =0;
	}
}

unsigned char Timer1_get_1sState(void)
{
	if (ucTimer1_1sFlag ==1)
	{
```

```
            ucTimer1_1sFlag = 0;
            return TIMER_TRIGGERED;
        }
        return TIMER_RUNNING;
}
```

- keyDeb.h

```
#ifndef KEYDEB_H_
#define KEYDEB_H_

#define KEYS_NOT_CHANGED 0
#define S1_PRESSED 1
#define S1_RELEASED 2
#define S2_PRESSED 3
#define S2_RELEASED 4

char keyInit();
char keyPressed(char key);
unsigned char key_get_state (void);

#endif
```

- keyDeb.c

```
#include <avr/io.h>
#include "keyDeb.h"

 unsigned char S1_new= 0xFF, S1_old = 0xFF;
 unsigned char S2_new= 0xFF, S2_old = 0xFF;

  char keyInit()
  {
        DDRD &= ~(1<<DDD2) & ~
        (1<<DDD3) & ~
        (1<<DDD4) & ~
        (1<<DDD5);
  }
  unsigned char key_get_state (void)
  {
       S1_old= S1_new;
       S1_new=PIND & (1<<PD5);
       if (!(S1_new) && (S1_old))
       {
            return S1_PRESSED;
       }
```

```
        else if (S1_new && !(S1_old))
        {
                return S1_RELEASED;
        }

        S2_old= S2_new;
        S2_new=PIND & (1<<PD4);
        if (!(S2_new) && (S2_old))
        {
         return S2_PRESSED;
        }
        else if (S2_new && !(S2_old))
        {
         return S2_RELEASED;
        }
        return KEYS_NOT_CHANGED;
}

 char keyPressed(char key)
{
        if ((key==1) && !(PIND & (1<<PD5))) return 1;
        else if ((key==2) && !(PIND & (1<<PD4))) return 1;
        else if ((key==3) && !(PIND & (1<<PD3))) return 1;
        else if ((key==4) && !(PIND & (1<<PD2))) return 1;
        else return 0;
}
```

- display_funktionen.h (reutilización de código)

```
#include <avr/io.h>
#include <avr/interrupt.h>

void Display_delay(unsigned long delay_time_us);

void Display_Aus(void);
void Display_An(void);
void Display_Init(void);

void Display_HardwareInit(void);

void Display_RS_Output(void);
void Display_RS_High(void);
void Display_RS_Low(void);

void Display_EN_Output(void);
void Display_EN_High(void);
void Display_EN_Low(void);
```

```
void Display_DATA_Output(void);
void Display_DATA_BitHigh(unsigned char DataBit);
void Display_DATA_BitLow(unsigned char DataBit);

void Display_Clear(void);
void Display_ReturnHome(void);
void Display_ModeEntry(unsigned char Options);
void Display_Control(unsigned char Options);
void Display_CursorOrDisplayShift(unsigned char Options);
void Display_SetMPUInterface(unsigned char Options);
void Display_SetCursor(unsigned char row, unsigned char column);
void Display_Transfer4BitData(unsigned char Data);
void Display_Write(unsigned char ASCII_of_char);
void Display_Print(unsigned char* text2print, unsigned char length);
void Display_GenerateNewChar(unsigned char address, unsigned char*
Pattern_New_Char);

#define DISPLAY_CLEAR_FUNCTION                  0x01
#define DISPLAY_CLEAR_DISPLAY_DELAY             4000

#define DISPLAY_RETURN_HOME_FUNCTION            0x02
#define DISPLAY_RETURN_HOME_DELAY               2000

#define DISPLAY_MODE_INCR_SHIFT_OFF             0x06
#define DISPLAY_MODE_INCR_SHIFT_ON              0x07
#define DISPLAY_MODE_DECR_SHIFT_OFF             0x04
#define DISPLAY_MODE_DECR_SHIFT_ON              0x05
#define DISPLAY_MODE_DELAY                      50

#define DISPLAY_OFF                             0x08
#define DISPLAY_ON_CURSOR_OFF                   0x0C
#define DISPLAY_ON_CURSOR_ON_BLINK_OFF          0x0E
#define DISPLAY_ON_CURSOR_ON_BLINK_ON           0x0F
#define DISPLAY_CONTROL_DELAY                   50

#define DISPLAY_SHIFT_CURSOR_RECHTS             0x14
#define DISPLAY_SHIFT_CURSOR_LINKS              0x10
#define DISPLAY_SHIFT_DISPLAY_RECHTS            0x1C
#define DISPLAY_SHIFT_DISPLAY_LINKS             0x18
#define DISPLAY_CURSOR_OR_DISPLAY_SHIFT_DELAY   50

#define DISPLAY_MPU_4BIT_2_LINES_5x7_DOTS       0x28
#define DISPLAY_SET_MPU_INTERFACE_DELAY         50

#define DISPLAY_FUNKTION_SET_DDRAM_ADRESSE      0x80
#define DISPLAY_FUNKTION_SET_CGRAM_ADRESSE      0x40
```

```
#define DISPLAY_SET_RAM_ADRESSE_DELAY		50

#define F_CPU 18432000UL

#define DISPLAY_SW_DDR_REG		DDRD
#define DISPLAY_SW_PORT_REG		PORTD
#define DISPLAY_SW_BIT			PD7

#define DISPLAY_RS_DDR_REG		DDRB
#define DISPLAY_RS_PORT_REG		PORTB
#define DISPLAY_RS_BIT			PB0

#define DISPLAY_EN_DDR_REG		DDRB
#define DISPLAY_EN_PORT_REG		PORTB
#define DISPLAY_EN_BIT			PB1

#define DISPLAY_DATA_DDR_REG		DDRC
#define DISPLAY_DATA_PORT_REG		PORTC
#define DISPLAY_DATA_DB7_BIT		PC3
#define DISPLAY_DATA_DB6_BIT		PC2
#define DISPLAY_DATA_DB5_BIT		PC1
#define DISPLAY_DATA_DB4_BIT		PC0

#define DATAPORTMASKE (~((1 << DISPLAY_DATA_DB7_BIT) | (1 << DISPLAY_DATA_DB6_BIT) | (1 << DISPLAY_DATA_DB5_BIT) | (1 << DISPLAY_DATA_DB4_BIT)))
```

- display_funktionen.c (reutilización de código)

```
#include "display_funktionen.h"


#define _1US_DELAY (20000000/F_CPU)

unsigned char ucDataBit[8] = {DISPLAY_DATA_DB7_BIT, DISPLAY_DATA_DB6_BIT, DISPLAY_DATA_DB5_BIT, DISPLAY_DATA_DB4_BIT, DISPLAY_DATA_DB7_BIT, DISPLAY_DATA_DB6_BIT, DISPLAY_DATA_DB5_BIT, DISPLAY_DATA_DB4_BIT};

unsigned char ucRAM_OffsetAdresse[4] = {0x00, 0x40, 0x10, 0x50};

void Display_HardwareInit(void)
{
	Display_DATA_Output();
	Display_RS_Output();
	Display_EN_Output();
```

```
}
void Display_delay(unsigned long delay_time_us)
{
	unsigned long delay_time;

	delay_time = delay_time_us / _1US_DELAY;
	for(unsigned long j = 0; j < delay_time; j++)
	{
	}
}
void Display_Aus(void)
{
	DISPLAY_SW_DDR_REG |= 1 << DISPLAY_SW_BIT;
	DISPLAY_SW_PORT_REG &= ~(1 << DISPLAY_SW_BIT);
}
void Display_An(void)
{
	DISPLAY_SW_PORT_REG |= 1 << DISPLAY_SW_BIT;
}
void Display_RS_Output(void)
{
	DISPLAY_RS_DDR_REG |= 1 << DISPLAY_RS_BIT;
}
void Display_RS_High(void)
{
	DISPLAY_RS_PORT_REG |= 1 << DISPLAY_RS_BIT;
}
void Display_RS_Low(void)
{
	DISPLAY_RS_PORT_REG &= ~(1 << DISPLAY_RS_BIT);
}
void Display_EN_Output(void)
{
	DISPLAY_EN_DDR_REG |= 1 << DISPLAY_EN_BIT;
}
void Display_EN_High(void)
{
	DISPLAY_EN_PORT_REG |= 1 << DISPLAY_EN_BIT;
}
void Display_EN_Low(void)
{
	DISPLAY_EN_PORT_REG &= ~(1 << DISPLAY_EN_BIT);
}
void Display_DATA_Output(void)
{
	DISPLAY_DATA_DDR_REG |= (1 << DISPLAY_DATA_DB7_BIT) | (1 <<
DISPLAY_DATA_DB6_BIT) |
```

```
                                        (1 << DISPLAY_DATA_DB5_BIT) | (1 <<
DISPLAY_DATA_DB4_BIT);
}
void Display_DATA_BitHigh(unsigned char DataBit)
{
        DISPLAY_DATA_PORT_REG |= (1 << DataBit);
}
void Display_DATA_BitLow(unsigned char DataBit)
{
        DISPLAY_DATA_PORT_REG &= ~(1 << DataBit);
}
void Display_Clear(void)
{
        Display_HardwareInit();
        Display_RS_Low();
        Display_Transfer4BitData(DISPLAY_CLEAR_FUNCTION);
        Display_RS_High();
        Display_delay(DISPLAY_CLEAR_DISPLAY_DELAY);
void Display_ReturnHome(void)
{
        Display_HardwareInit();
        Display_RS_Low();
        Display_Transfer4BitData(DISPLAY_CLEAR_FUNCTION);
        Display_RS_High();
        Display_delay(DISPLAY_RETURN_HOME_DELAY);
}
void Display_ModeEntry(unsigned char Options)
{
        Display_HardwareInit();
        Display_RS_Low();
        Display_Transfer4BitData(Options);
        Display_RS_High();
        Display_delay(DISPLAY_MODE_DELAY);
}
void Display_Control(unsigned char Options)
{
        Display_HardwareInit();
        Display_RS_Low();
        Display_Transfer4BitData(Options);
        Display_RS_High();
        Display_delay(DISPLAY_CONTROL_DELAY);
}
void Display_CursorOrDisplayShift(unsigned char Options)
{
        Display_HardwareInit();
        Display_RS_Low();
        Display_Transfer4BitData(Options);
```

```
	Display_RS_High();
	Display_delay(DISPLAY_CURSOR_OR_DISPLAY_SHIFT_DELAY);
}
void Display_SetMPUInterface(unsigned char Options)
{
	unsigned char dummy = 0x80, i;
	Display_HardwareInit();
	Display_RS_Low();
	Display_EN_High();

	for(i = 0; i < 4; i++)
	{
		if(Options & dummy)
		{
			Display_DATA_BitHigh(ucDataBit[i]);
		}
		else
		{
			Display_DATA_BitLow(ucDataBit[i]);
		}

		dummy = dummy >> 1;
	}

	Display_EN_Low();
	Display_EN_High();
	Display_Transfer4BitData(Options);
	Display_RS_High();
	Display_delay(DISPLAY_SET_MPU_INTERFACE_DELAY);
}
void Display_SetCursor(unsigned char row, unsigned char column)
{
	unsigned char adresse = DISPLAY_FUNKTION_SET_DDRAM_ADRESSE;

	adresse += ucRAM_OffsetAdresse[row] + column;

	Display_HardwareInit();
	Display_RS_Low();
	Display_Transfer4BitData(adresse);
	Display_RS_High();
	Display_delay(DISPLAY_SET_RAM_ADRESSE_DELAY);
}
void Display_Transfer4BitData(unsigned char _8BitData)
{
	unsigned char dummy = 0x80, i;

	Display_EN_High();
```

```
        for(i = 0; i < 8; i++)
        {
                if(_8BitData & dummy)
                {
                        Display_DATA_BitHigh(ucDataBit[i]);
                }
                else
                {
                        Display_DATA_BitLow(ucDataBit[i]);
                }

                dummy = dummy >> 1;
                if(i == 3)
                {
                        Display_EN_Low();
                        Display_EN_High();
                }
        }

        Display_EN_Low();
}
void Display_Init(void)
{
        Display_Aus();
        Display_delay(50000);
        Display_An();
        Display_delay(30000);
        Display_SetMPUInterface(DISPLAY_MPU_4BIT_2_LINES_5x7_DOTS);
        Display_Clear();
        Display_ModeEntry(DISPLAY_MODE_INCR_SHIFT_OFF);
        Display_Control(DISPLAY_ON_CURSOR_ON_BLINK_OFF);
}
void Display_Write(unsigned char ASCII_of_char)
{
        Display_HardwareInit();
        Display_RS_High();
        Display_Transfer4BitData(ASCII_of_char);
        Display_RS_Low();
        Display_delay(DISPLAY_SET_RAM_ADRESSE_DELAY);
}
void Display_Print(unsigned char* text2print, unsigned char length)
{
        unsigned char i = 0;

        Display_HardwareInit();
        Display_RS_High();
```

```
	for(i = 0; i < length; i++)
	{
		Display_Write(text2print[i]);
	}
	Display_RS_Low();
}

void Display_GenerateNewChar(unsigned char address, unsigned char* Pattern_New_Char)
{
	unsigned char adresse = DISPLAY_FUNKTION_SET_CGRAM_ADRESSE;
	unsigned char i;

	adresse |= address << 3;
	Display_HardwareInit();
	Display_RS_Low();
	Display_Transfer4BitData(adresse);
	Display_RS_High();
	Display_delay(DISPLAY_SET_RAM_ADRESSE_DELAY);

	for(i = 0; i < 8; i++)
	{
		Display_Write(Pattern_New_Char[i]);
	}
	Display_RS_Low();
}
```