Práctica 4
Introducción a ATMEL Studio

Islas Bravo, Andrés.
Estudiante - Laboratorio de Microcontroladores.
Departamento de Arquitectura e Ingeniería
Instituto Tecnológico y de Estudios Superiores de Monterrey
Ciudad de México, México
a01339391@itesm.mx

Valverde López, Juan Pablo.
Estudiante - Laboratorio de Microcontroladores.
Departamento de Arquitectura e Ingeniería
Instituto Tecnológico y de Estudios Superiores de Monterrey
Ciudad de México, México
a01656127@itesm.mx

Zhu Chen, Alfredo.
Estudiante - Laboratorio de Microcontroladores.
Departamento de Arquitectura e Ingeniería
Instituto Tecnológico y de Estudios Superiores de Monterrey
Ciudad de México, México
a01651980@itesm.mx

Abstract
This practice consists in the appliance of Proteus knowledge to simulate an ATMega16 working. The new concepts and software to be introduced is ATMEL Studio and assembler code. The code is evolving basic programming level in order to get a first familiarisation with the environment and its properties. The data of the code into the ATMega16 will be loaded before simulation in order to make it easier.

I. Nomenclatura

memoria de acceso aleatorio,	*RAM.*
Unidad de microcontrolador,	*MCU.*
Entorno de desarrollo Integrado,	*IDE.*
Resistencia-Capacitor,	*RC.*
Unidad de frecuencia, hercio,	*Hz.*

II. Introducción

Dentro del presente documento se sintetiza lo realizado sobre la plataforma de ATMEL Studio. Se muestran los aspectos realizados durante la práctica que van desde la parte de hardware, correspondiente en este caso al atmega16 y sus periféricos como los leds, hasta la parte de código ensamblador el cual fue cargado al atmega16 simulado con la finalidad de tener una percepción realista de lo que sucedería en uno real. Será especial énfasis en la parte del código ya que durante esta práctica fue lo que se introdujo como información nueva, es por ello que inclusive dicho código tendrá una muy detallada señalización sobre lo sucedido en cada línea. Entre los funcionamientos esperados será un corrimiento de LEDs de 4 filas, esto mediante la señalización lógica en los puertos.

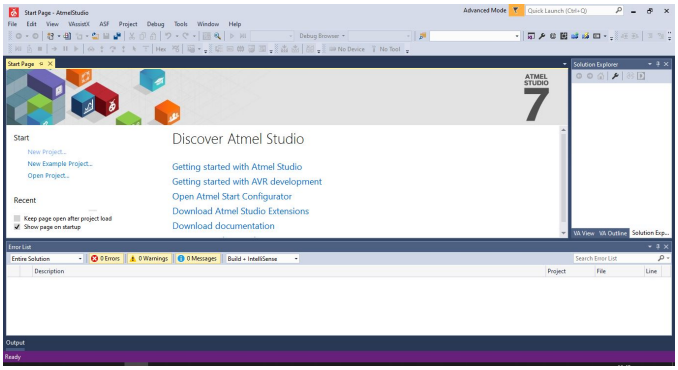
III. Marco Teórico

ATMEL Studio en su versión 7 es una plataforma de desarrollo integrado para desarrollar y depurar aplicaciones de microcontroladores AVR y SAM de la familia de *Microchip Technology*. Este mismo permite escribir, crear y depurar códigos en lenguajes C/C + + y ensamblador. Así mismo, tiene una gran compatibilidad con kits de desarrollo que soportan más de 500 dispositivos de la familia AVR y SAM[1].

El lenguaje ensamblador es un lenguaje de programación de bajo nivel y se programa directo sobre el *Hardware*. Esto permite tener más control sobre el dispositivo en el que se programa y más capacidad de velocidad junto con menor consumo de memoria. Cabe mencionar que es difícilmente portable programar en bajo nivel, ya que está restringido generalmente por las especificaciones de hardware por el fabricante[2].

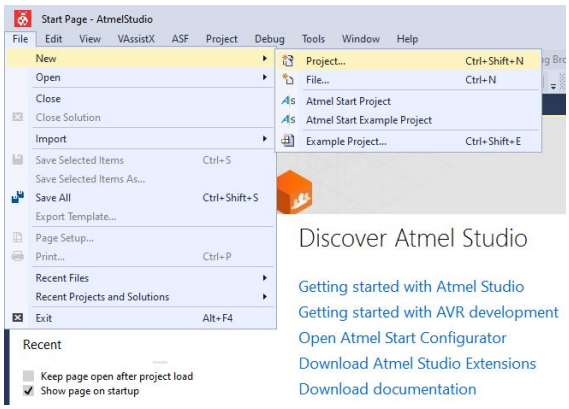
IV. Desarrollo

La presente práctica pretende introducir el uso de ATMEL Studio, que es un entorno de desarrollo integrado (IDE por sus siglas en inglés) para la programación y simulación de microcontroladores ATMEL. Dentro de las exigencias y temas que pretende responder este curso, se trabajará con el microcontrolador ATMega16. Por lo que el proyecto dentro del IDE, se inicia como:



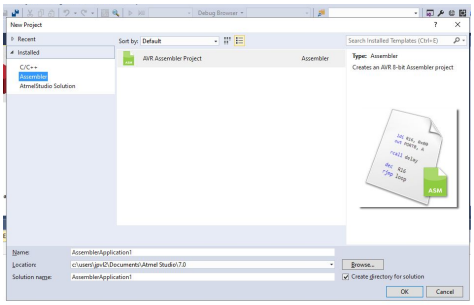
Captura 1: Pantalla de Bienvenida ATMEL Studio.

Para iniciar un nuevo proyecto, en la pestaña “File”, luego “New” y al final “Project”. O al presionar las tres teclas “Ctrl+Shift+N” al mismo tiempo.



Captura 2: Creación de un nuevo proyecto.

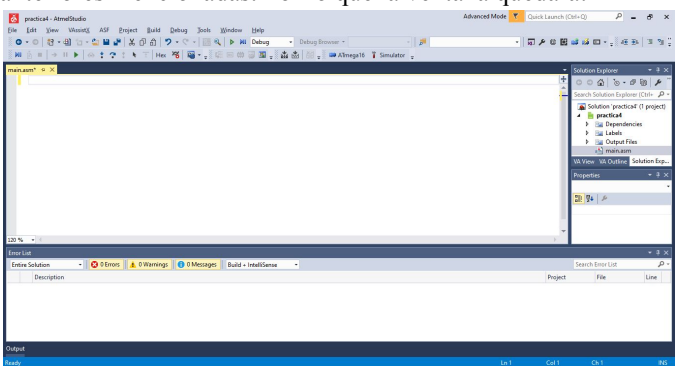
Luego de haber seleccionado la opción de crear un nuevo proyecto, una ventana de asistencia emergerá, en la que debemos elegir la opción de "Assembler" por sobre la C/C++ o Atmel Studio Solution, pues se pretende programar el microcontrolador en lenguaje ensamblador. Dentro de la misma ventana, también se escoge el directorio donde se guardará el proyecto así como el nombre bajo el cual se podrá identificar el mismo.



Captura 3: Ventana de Asistencia, Creación de Proyecto. Lenguaje Ensamblador.

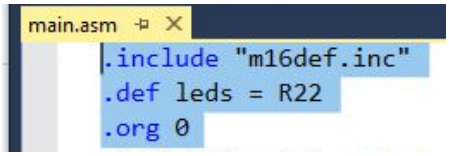
También se debe buscar el modelo del microcontrolador a utilizar, en este caso será el "ATMEGA16", es importante recalcar puesto que una versión muy similar es el "ATMEGA16A" pero este no se ocupará dentro del curso. En realidad este aspecto es a consideración del programador pues depende de cuál tenga disponible físicamente.

Una vez elegidos los detalles anteriores, una ventana emergerá con nombre "main.asm" la cual llevará escrita una plantilla modelo para poder programar con cierta guía. Sin embargo, para esta y futuras ocasiones, se eliminarán las líneas anteriores mencionadas. Por lo que la ventana quedará:



Captura 4: Ventana "main.asm".

Las primeras "instrucciones" serán para el compilador, son directrices para funcionar de cierta manera. En este caso las 2 instrucciones resaltadas significan:



Captura 5: Instrucciones Directrices para el compilador.

Instrucción - Directriz	Interpretación
.include "*mcu*.inc"	Incluye las instrucciones y comandos, memorias, relacionadas con el (m)ega(16)(def)initions
.def *name* = *register*	Define a un registro con una etiqueta de caracteres que el programador elija
.org #	Empieza a escribir desde la dirección de memoria flash # en adelante.

Tabla 1: Instrucciones directrices para el compilador.

El programa pretende funcionar como el corrimiento del encendido de LEDs, para ello se utilizaron los puertos A y B como salida. Las instrucciones y su función se encuentran comentadas línea por línea dentro de las siguientes capturas de pantalla.

```
main.asm*
;instrucciones para el compilador
.def leds = R22;renombrando el registo r22 como "leds", definiéndolo
.include "m16def.inc";instruccion al compilador, traducción a código máquina del atmega16
.org 0 ;directiva que le dice al compilador "a partir de aquí cualquier instruccion
;la vas a guardar a partir de la dirección 0 de memoria flash"

;Definicion del puntero, siempre se hace
LDI R16, HIGH (RAMEND);última dirección de memoria ram
OUT SPH, R16
LDI R16, LOW (RAMEND)
OUT SPL, R16
;Configuracion puntero RAM
LDI XL, LOW(0x0060);8 bits, en este caso "60"
LDI XH, HIGH(0x0060);8 bits, en este caso "00"
LDI R16, 10;cargando a r16 un 10 decimal
LDI R17, 0x22;cargando un 22 hexa
MOV R24, R17;movimiento r17 a r24 para probar funciones
;total 10 saltos
```

Captura 6: Código para el encendido-corrido de LEDs.

```
main.asm*
loop:
ST X+, R17;guardar en RAM en r17 la dirección a la que apunta X
DEC R16;decrementado r16 de 10 a 9
BRNE loop;branch (salto) not equal. Salta si no es igual (IGUAL A 0), quién, lo último que hubo, en este caso compara r16
;z se pone a 1 cuando algún resultado de una instrucción sea 0. INC es una instrucción de la ALU y pregunta
;constantemente por la bandera Z

;Hasta aqui memoria RAM
//
SER R16;SEt Register, setea a 1 el R16
OUT DDRA, R16;1 salida ddra vale 1 los 8 bits, los 8 bits de ddra son salidas
LDI R23, 8
CLR leds; leds=r22
SEC     ;setea el Carry=1
```

Captura 6.1: Código para el encendido-corrido de LEDs.

```
izq:
ROL leds;corrimiento ROll Left, giro a la izquierda
;hace un corrimiento del carry y el último dígito desplazado va al carry
OUT PORTA, leds;port escribir pin leer
CALL delay100ms;salta a la etiqueta
DEC R23
BRNE izq;ejecutada 8 veces
LDI R23, 8

der:
ROR leds;corrimiento Roll Right, carry entra del lado izquierd
OUT PORTA, leds
CALL delay100ms;salta pero regresa a donde fuiste llamado
DEC R23
BRNE der
```

Captura 6.2: Código para el encendido-corrido de LEDs.

```
FIN:
RJMP FIN; relative jump. Hay que terminar obligatoriamente en este loop

delay10ms:  LDI R19, 104
    loop1:  LDI R20, 255
    loop2:  DEC R20
            BRNE loop2
            DEC R19
            BRNE loop1
RET;RETurn regresa de donde fue llamado MUY IMPORTANTE
;cómo sabe a dónde regresar, por el puntero de la pila
;el RET trata de extraer la dirección de la pila
delay100ms:
        LDI R21, 10
loop3:  CALL delay10ms
        DEC R21
        BRNE loop3
RET
```

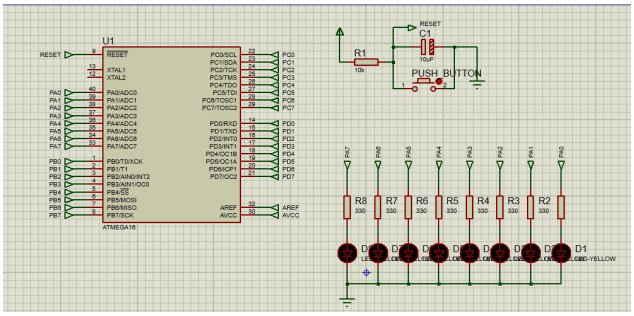
Captura 6.3: Código para el encendido-corrido de LEDs.

En principio, como es descrito en las líneas de código que integran este programa, el corrimiento empieza de izquierda a derecha y luego se detiene. Esto puede ser comprobado de dos formas, la primera desde el programa auxiliar Proteus en su versión 8.8, donde se carga el archivo .hex que es generado en el directorio del proyecto. Además una segunda forma, algo más técnica, es que una vez compilado el programa, la ejecución por cada línea de código otorga la libertad de inspeccionar el accionar del microcontrolador por cada ejecución de instrucción. Esta es una forma de detección de programas que exige un mayor conocimiento del sistema donde se está trabajando y la solución e interpretación para poder llevar a cabo la tarea designada.

Alternativa 1: Comprobación por Proteus.

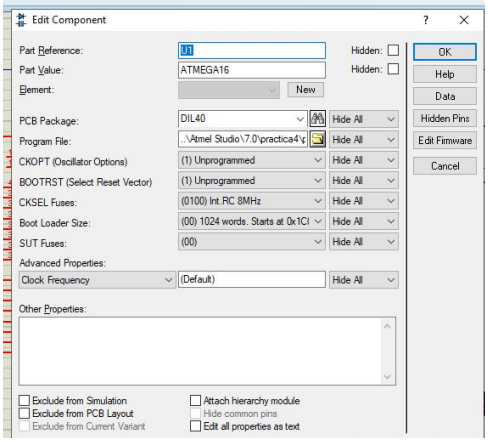
Para esta comprobación se hará uso del programa anterior diseñado, el sistema mínimo del ATMega16, que básicamente consiste en el etiquetado y programación de un botón de RESET para el MCU.

Las salidas en este caso, serán 8 LEDs que están conectadas con sus respectivas resistencias y aterrizadas a un extremo de sus pines.



Captura 7: Sistema Mínimo del ATMega16 en conexión a sus salidas LEDs.

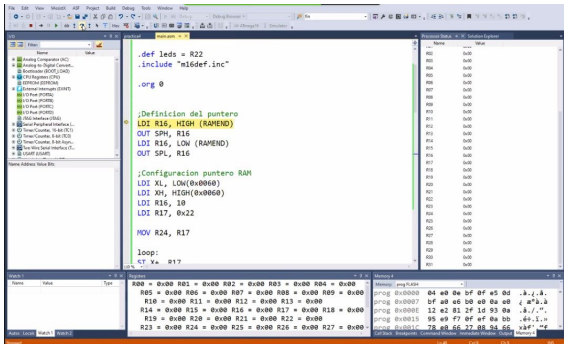
Para cargar el archivo de instrucción de código máquina al MCU, se da click derecho sobre él y se selecciona en "Edit Properties". Donde emergerá una ventana, allí en "Program File" se carga la dirección del programa con extensión .hex de donde se servirá el MCU para simular el comportamiento.



Captura 8: Carga de archivo ".hex" para la simulación de corrimiento de LEDs.

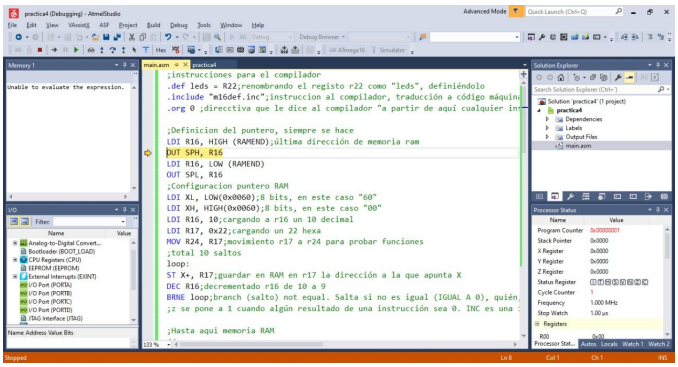
Para la selección del la frecuencia a la que trabajará el MCU, se lo hace en el apartado de "CKSEL Fuses" que prepara al MCU a trabajar de cierta forma, en este caso específico "0100", lo que significa que funcionará con su reloj interior, hecho con un arreglo de RC a 8MHz.

Alternativa 2: Comprobación dentro del IDE Atmel Studio.



Captura 9: Compilación del código para el encendido-corrido de LEDs.

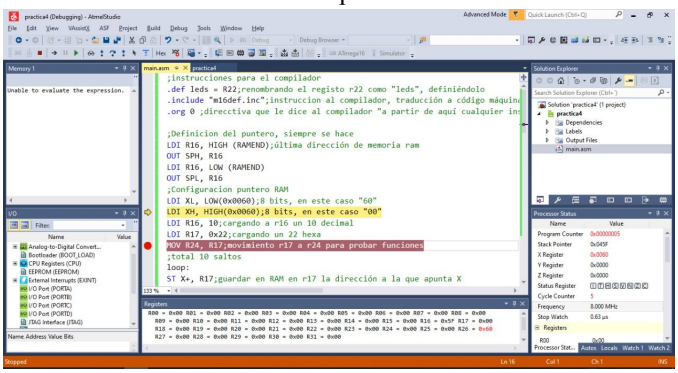
Una vez compilado el programa la ejecución empieza desde la primera línea después de las directrices al compilador, en este caso desde las cuatro instrucciones a partir de "LoaD Immediate"(seleccionado en amarillo) que prepara el puntero a pila (paso fundamental para configurar la dirección del apuntador de pila). Se configura la dirección del apuntador de pila a la última parte de la memoria RAM(RAMEND)



Captura 10: Ejecución de la siguiente instrucción del código para el encendido-corrido de LEDs.

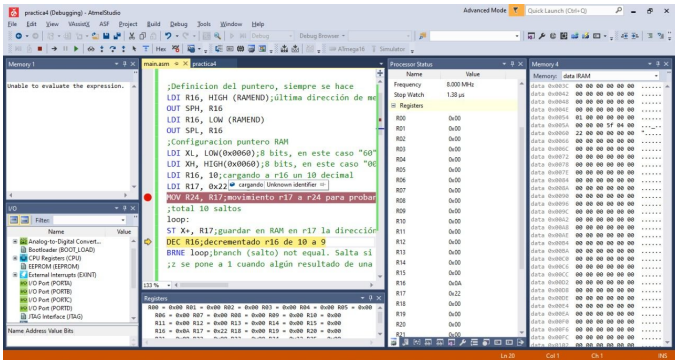
Para ejecutar las demás instrucciones, se lo hace con el ícono de flecha apuntando hacia la derecha, que se encuentra al lado derecho del ícono cuadrado rojo que significa detener la ejecución del programa. Es importante mencionar que la ejecución es secuencial, es decir una tras otra.

Algunas de las instrucciones pueden tornarse repetitivas cuando se sabe qué es lo que está haciendo el MCU, por ello podemos formular etiquetas llamadas "breakpoints" resaltando la línea donde se quiere hacer un salto de instrucción y dando un click en el costado izquierdo de la franja verde, donde aparecerá un círculo rojo. El programa ejecutará las instrucciones sin detenerse una por una hasta una línea antes de donde se encuentra el "breakpoint".



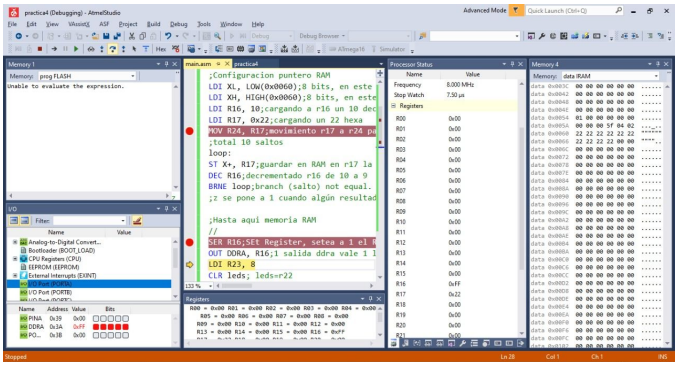
Captura 11: Breakpoints en instrucciones del código para el encendido-corrido de LEDs.

Es importante llevar un recuento sobre los registros que se modifican instrucción a instrucción, para ello existen varios recuadros que detallan la actividad de múltiples espacios del MCU.



Captura 12: Ventanas de funcionalidad a atender por el programador dentro del código para el encendido-corrido de LEDs.

En la captura adjunta se puede apreciar cómo el puerto de entrada/salida "i/o" se altera con las instrucciones cargadas.



Captura 13: Comprobación de alteración en puerto de i/o.

Como actividad requisito y práctica de la lógica de programa se pidió que se activaran todos los leds por los 4 puertos del MCU, y que el corrimiento sea infinito. El programa en esencia se mantiene, aunque ciertos cambios importantes fueron realizados para que cumpla con la cometida.

```
main.asm  main.asm*
    .include "m16def.inc"
    .org 0
    ;Definicion del puntero
    LDI R16, HIGH (RAMEND);ox45f
    OUT SPH, R16
    LDI R16, LOW (RAMEND)
    OUT SPL, R16

    SER R16
    SER R17
    SER R18
    SER R19
    OUT DDRA, R16
    OUT DDRB, R17
    OUT DDRC, R18
    OUT DDRD, R19
    FIN:
```

Captura 14: Programa que enciende LEDs en los cuatro puertos y los corre de izquierda a derecha, luego de derecha a izquierda de manera infinita.

```
main.asm  main.asm*
    LDI R16, 8
    CLR R4
    SEC; C=1
    izq:
    ROL R4
    OUT PORTA, R4
    OUT PORTB, R4
    OUT PORTC, R4
    OUT PORTD, R4
    CALL delay100ms
    DEC R16
    BRNE izq
    LDI R16, 8
```

Captura 14.1: Programa que enciende LEDs en los cuatro puertos y los corre de izquierda a derecha, luego de derecha a izquierda de manera infinita.

```
der:
ROR R4
OUT PORTA, R4
OUT PORTB, R4
OUT PORTC, R4
OUT PORTD, R4
CALL delay100ms
DEC R16
BRNE der
RJMP FIN
```

Captura 14.2: Programa que enciende LEDs en los cuatro puertos y los corre de izquierda a derecha, luego de derecha a izquierda de manera infinita.

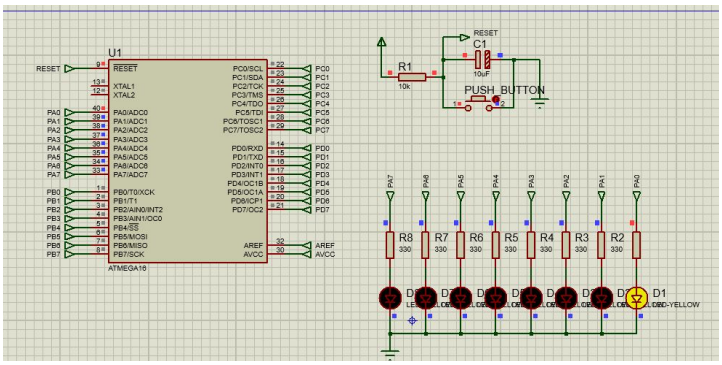
```
delay10ms:  LDI R20, 104
    loop1:  LDI R21 ,255
    loop2:  DEC R21
            BRNE loop2
            DEC R19
            BRNE loop1
RET
delay100ms:
        LDI R22, 10
loop3:  CALL delay10ms
        DEC R22
        BRNE loop3
RET
```

Captura 14.3: Programa que enciende LEDs en los cuatro puertos y los corre de izquierda a derecha, luego de derecha a izquierda de manera infinita.

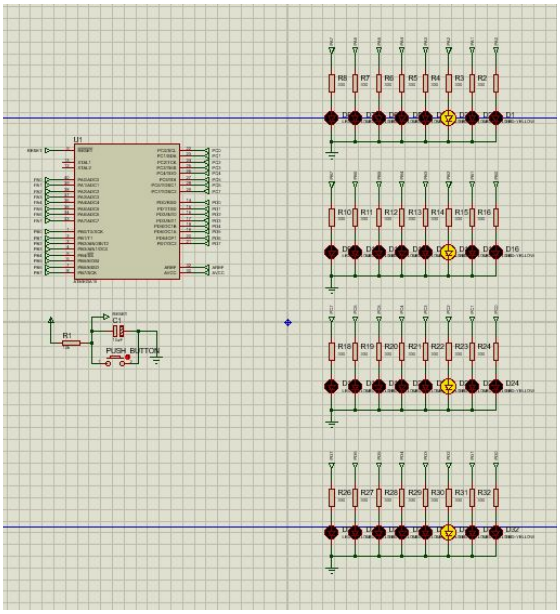
Como diferencia fundamental se denota la inicialización de todos los puertos así como el uso de la etiqueta "FIN:" al inicio del programa y su instrucción "RJMP" seguida del nombre antes mencionado, lo que provoca esto es que el programa se ejecuta una y otra vez hasta que sea reseteado y vuelva a ejecutar su código, entra en un loop infinito porque después de ejecutar el corrimiento de izquierda a derecha es llamado al inicio para que lo vuelva a hacer.

V. Resultados

A. Corrimiento de LEDs finito de izquierda a derecha



Captura 15 : Inicio del corrimiento de LEDs Finito..



Captura 16 : Segunda acción del corrimiento de LEDs infinito.

VI. Conclusiones

Andrés Islas Bravo:

Esta práctica permitió la correcta y primera familiarización con el ambiente desarrollo atmel Studio 7, así como su directa relación con los controladores atmega16. Al ser la primera práctica realizada con Código ensamblador se aprendieron nuevos conceptos referentes al lenguaje. Dentro de estos conceptos aprendidos se encuentran la correcta estructuración de la pila cómo los punteros, así mismo se entendió que realmente nunca se deja de correr el código, sino que realmente se configura un salto sin condición a una línea previa generando un Loop infinito. Finalmente, un aspecto muy importante de atmel Studio 7 es la cuestión de depuración de un programa, esto para eliminar errores previos a la ejecución.

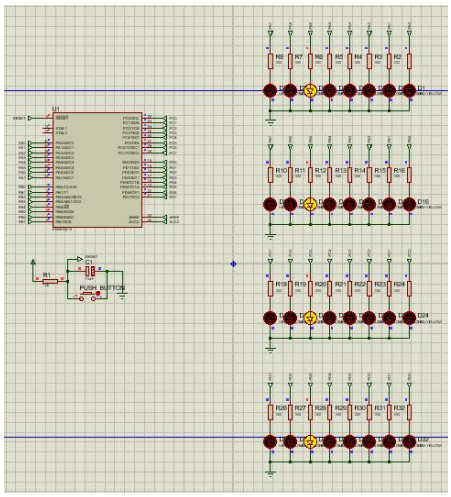
Juan Pablo Valverde López:

Es de suma importancia mantener un conocimiento técnico y estructural del microcontrolador que se esté utilizando, el IDE ofrece la posibilidad de evaluar qué está haciendo el MCU por cada ejecución de instrucción que hace y tener un máximo control sobre los problemas que llegue a experimentar. La misma arquitectura del lenguaje, que es secuencial ofrece de

primera vista ventajas por sobre la depuración de programas.

Alfredo Zhu Chen:

Esta práctica me permitió conocer el ambiente de desarrollo de ATMEL Studio 7. En la plataforma logré simular el código de la práctica y observar los efectos que tiene sobre las partes físicas del Atmega16. El IDE es una herramienta muy útil para escribir y depurar los códigos, correr los códigos por cada paso y mediante el uso de *breakpoints* permite tener conocimiento de lo que está sucediendo en el programa. Así mismo, fue interesante poder probar el código escrito en el programa de simulación de Proteus para observar el funcionamiento de manera visual.



VII. Referencias

[1] Microchip. (s.f.). Atmel Studio 7. Obtenido desde: https://www.microchip.com/mplab/avr-support/atmel-studio-7

[2] Britton, R. (2004). MIPS Assembly Language Programming. Prentice Hall.

VIII. Apéndices