Práctica 5
Puertos de Entrada y Salida del ATMega 16

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 elaboration of multiple circuits, each one with a particular function. Among the tasks covered by those circuits are counting, rolling leds and displaying a value in a seven segments display. The code was done over an atmel Studio environment and the ideas are structured in a flowchart diagram. There is also a hierarchy order which defines which switch is going to have the last word to decide the actions that are going to be made.

I. Nomenclatura

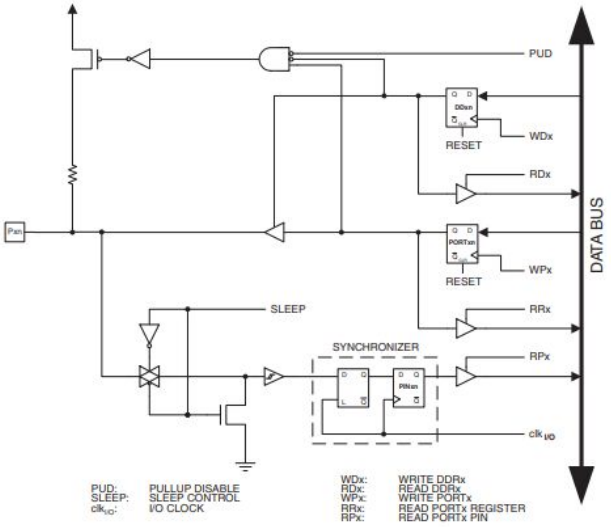
memoria de acceso aleatorio,	*RAM.*
Unidad de microcontrolador,	*MCU.*
Resistencia-Capacitor,	*RC.*
Unidad de frecuencia, hercio,	*Hz.*
Diodo Emisor de Luz,	*LED.*

II. Introducción

Dentro del presente documento se abordan aspectos tanto de programación, como la estructura del Hardware, así como una estructuración del pensamiento para la realización de múltiples tareas. En los aspectos de programación se siguió manejando un entorno de atmel Studio el cual permite la simulación y análisis de un programa, refiriendo a la estructura del Hardware se realizó en proteus, programa el cual permite la visualización del funcionamiento realista de los componentes conectados. Como un aspecto agregado en esta práctica se hizo uso de diagramas de flujo los cuales permitieron una previsualización de cómo sería estructurado el código para cumplir las tareas deseadas, ya que estas tenían un orden jerárquico dependiendo del switch al que fueran asignadas.

III. Marco Teórico

El manejo de los puertos de entrada y salida es esencial para el microcontrolador, ya que mediante ellos se envían o reciben los datos de información y también pueden tener funciones especiales del mismo microcontrolador. ATMega16 cuenta con 4 puertos de 8 bits, cada uno consiste en 3 registros de 8 bits: DDx,PORTx, PINx. DDxn es el registro que se utiliza para configurar el puerto en modo de entrada o de salida, un uno lógico en el bit indica que trabajará como salida y un cero lógico como entrada. Cabe mencionar que los puertos del microcontrolador ATmega16 arrancan como modo de entrada, es decir todos los bits de DDx en cero lógico. Así mismo, PORTx se utiliza para escribir los datos que se quieren sacar del puerto y PINx para leer si son datos de entrada. Como se muestra en la captura 1, los puertos contienen una resistencia interna de *pull up* para cuando se utilice como modo entrada[1].



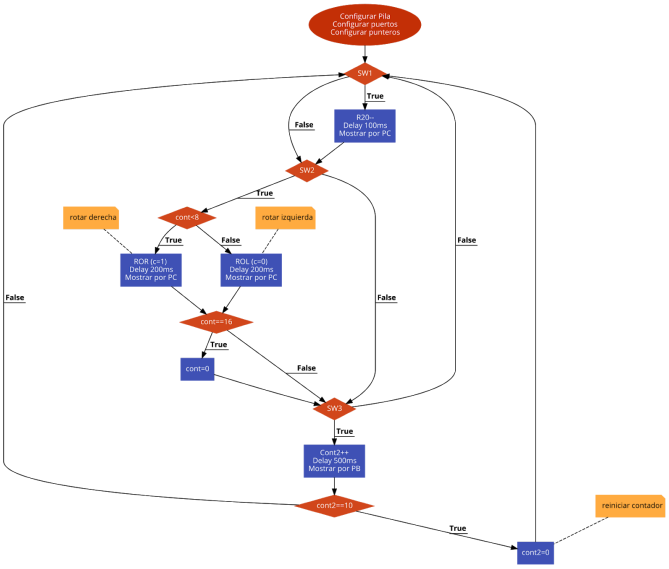
Captura 1: Puertos de E/S en ATMega16

IV. Desarrollo

La presente práctica pretende hacer uso de 3 puertos del ATMega16, que serán configurados como salidas y así puedan realizar una tarea específica. La aplicación del ATMega permitirá leer el estado de 3 switches, dependiendo de cuál esté activo se realizará una secuencia con LEDs.

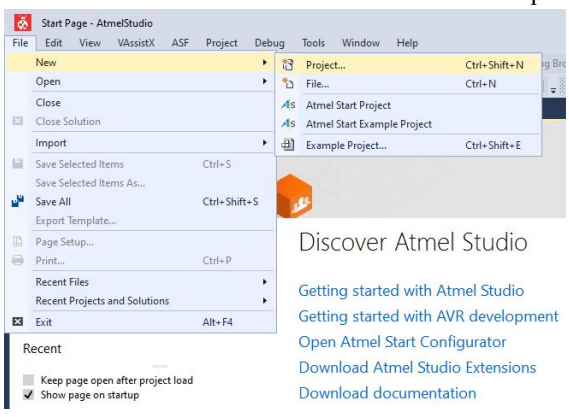
- **Secuencia A:** Iniciar un conteo descendente de 8 bits. 255, 254, 253..., 1, 0, 255, 254...(a una velocidad de 100 ms entre cada paso) y mostrar por el puerto C.
- **Secuencia B:** Hacer una secuencia donde 8 LEDs se vayan encendiendo de izquierda a derecha hasta que todos estén encendidos. Después irlos apagando de uno en uno de derecha a izquierda. (a una velocidad de 200 ms entre cada paso). Mostrarlo por el puerto C.
- **Secuencia C:** En un display 7 segmentos, hacer un conteo ascendente. (a una velocidad de 500 ms entre cada paso). Mostrarlo por el puerto B.

Como recomendación para el presente y futuros desarrollos de este tipo de programa, se realizará un pseudocódigo a manera de diagrama de flujo que explicará a grandes rasgos el funcionamiento del programa.



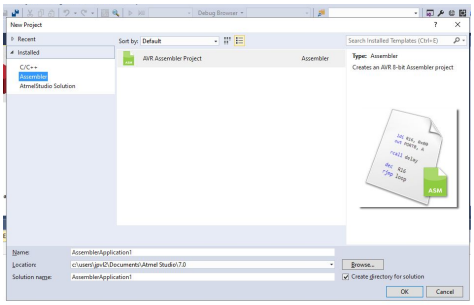
Captura 2: Diagrama de Flujo.

Para iniciar un nuevo proyecto, es necesario dirigirse a la pestaña "File", luego "New" y al final "Project". O al presionar las tres teclas "Ctrl+Shift+N" al mismo tiempo.



Captura 3: Creación de un nuevo proyecto.

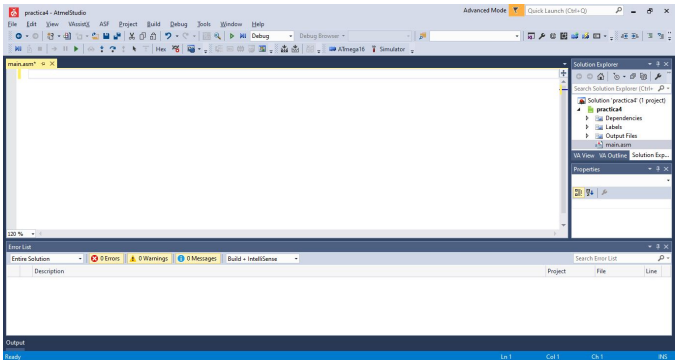
Luego de haber seleccionado la opción de crear un nuevo proyecto, una ventana de asistencia emergerá, en la que debemos elegir la opción de "Assembler" por sobre la C/C++ o Atmel Studio Solution, pues se pretende programar el microcontrolador en lenguaje ensamblador. Dentro de la misma ventana, también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



Captura 4: Ventana de Asistencia, Creación de Proyecto. Lenguaje Ensamblador.

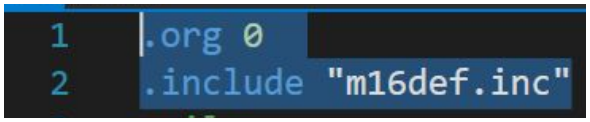
También se debe buscar el modelo del microcontrolador a utilizar, en este caso será el "ATMEGA16", es importante recalcar puesto que una versión muy similar es el "ATMEGA16A" pero este no se ocupará dentro del curso. En realidad este aspecto es a consideración del programador pues depende de cuál tenga 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 5: 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 6: 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

.org #	Empieza a escribir desde la dirección de memoria flash # en adelante.

Tabla 1: Instrucciones directrices para el compilador.

Es necesario preparar la pila que hará uso de las últimas direcciones de la memoria RAM. Si bien es cierto que la pila no es ocupada en este programa en específico, sirve para guardar la dirección de retorno de una interrupción o subrutina además de guardar datos importantes ocupados mientras se estaba ejecutando otras instrucciones de tu programa. (PUSH/POP).

Una buena práctica en este tipo de programas es comentar aquellas características más importantes del problema.

```
/*
programa que lea el estado de 3 switches, dependiendo cuál esté activo realizar
   1) Iniciar un conteo descendente de 8 bits. 255, 254, 253.., 1, 0
   , 255, 254...(100ms entre cada paso).
   Mostrar por el puerto C.
   2) Secuencia donde 8 LEDs se vayan encendiendo de izquierda a derecha
   hasta que todos estén encendidos.
   Irlos apagando de uno en uno derecha a izquierda.
   (Velocidad de 200ms entre cada paso).
   Mostrarlo por el puerto C.
   3) En un display 7 segmentos, hacer un conteo ascendente.
   (Velocidad de 500ms entre cada paso).
    Mostrarlo por el puerto B.
    Nota: Dos switches no pueden estar en ON al mismo tiempo
*/
```

Captura 7: Resumen características importantes del problema a realizar..

Como ya se mencionó anteriormente, el programa pretende funcionar de cierta manera si uno de sus 3 switches son accionados. La secuencia de acciones a realizar por cada switch son sensibles a si su directriz (switch accionado) sigue accionado luego de haber concluido su tarea. Para mayor claridad, cada línea de código está comentada. Como primer paso se tiene que preparar los puertos necesarios como salida.

```
;preparar puertos como salida
ser r16 ;seteo registro r16(lleno de 1s)
out ddrc, r16;puerto c como salida
out ddrb, r16;puerto b como salida
```

Captura 8: Código - Preparación de puertos.

```
pollingswitch:;etiqueda para sensado de switches
    sbic pind,0;switch 1
    rjmp pollingswitch2
    ldi r16, 255;cargo 255 a r16 (contador)
retorno0:
;vigilar switches
    sbic pind,0;si levantas el switch 1 a media operación sáltate al sw2
    rjmp pollingswitch2
    sbis pind,1;switch 2
    rjmp pollingswitch2;si aprietas el switch 2 a media operación sáltate al sw3
    sbis pind,2;switch 3
    rjmp pollingswitch3
    out portc, r16;muestro mi contador por el puerto C
    call delay100ms;hago el delay
    dec r16;decremento contador
    cpi r16, 0;pregunto si el contador llega a 0
```

Captura 8.1: Código - Sensado de Switch 1.

```
    brne retorno0;salta si no es igual (z)
    out portc, r16
    rjmp pollingswitch
pollingswitch2:
    sbic pind,1;switch 2
    rjmp pollingswitch3
    ldi r16, 0;cargo 0 a r16 (barrido)
    ldi r17, 0;cargo un 0 a un contador
retorno1:
;vigilar switches
    sbis pind,0;switch 1, si aprietas el SW1 a media operacion salta a SW1
    rjmp pollingswitch
    sbic pind,1;switch 2
    rjmp pollingswitch3;si levantas el switch 2 a media operación sáltate al sw3
    sbis pind,2;switch 3
    rjmp pollingswitch3
```

Captura 8.2: Código - Sensado de Switch 2.

```
    rjmp pollingswitch3
    sec ;seteo carry para vigilarlo en las rotaciones
    call delay200ms
    ror r16;(barrido) roto hacia la derecha r16(inicialmente en 0)
    out portc, r16;muestro mi barrido por el puerto C
    inc r17;incremento contador
    cpi r17, 8;pregunto si el conteo es 8
    brne retorno1;salta si no es igual (z)
    ;r17 es 8
    clr r17;conteo a 0
retorno2:
;vigilar switches
    sbis pind,0;si aprietas el SW1 a media operacion salta a SW1
    rjmp pollingswitch
    sbic pind,1;switch 2
    rjmp pollingswitch3;si levantas el switch 2 a media operación sáltate al sw3
```

Captura 8.3: Código - Retornos Switch 1 y 2.

```
    clc ;clear carry
    call delay200ms
    rol r16;(barrido) a la izquierda
    out portc, r16
    inc r17
    cpi r17, 8
    brne retorno2;salta si no es igual (z)
    rjmp pollingswitch
pollingswitch3:
    sbic pind,2;switch 3
    rjmp pollingswitch
    ldi r16, 0;cargo 0 a r16 (display)
retorno3:
;vigilar switches
    sbis pind,0;si aprietas el SW1 a media operacion salta a SW1
    rjmp pollingswitch
```

Captura 8.4: Código - Sensado de Switch 3.

```
    sbis pind,1;si aprietas el SW2 a media operacion salta a SW2
    rjmp pollingswitch2;
    sbic pind,2;switch 3
    rjmp pollingswitch;si levantas el sw3 a media operacion sáltate al sw1
    out portb, r16;muestro mi contador por el puerto b
    call delay500ms;hago el delay 500ms
    inc r16;incremento contador
    cpi r16, 10;pregunto si el contador llega a 10 (quiero que se visualice el 9)
    brne retorno3;salta si no es igual (z)
    rjmp pollingswitch
delay10ms: ldi r20,104
    ciclo2:
        ldi r21, 255
    ciclo1:
        dec r21
        brne ciclo1
```

Captura 8.5: Código - Sensado de Switch 3. Delay 1

```
        dec r20
        brne ciclo2
ret
delay100ms: ldi r22, 10
    ciclo3:
        call delay10ms
        dec r22
        brne ciclo3
ret
delay200ms: ldi r23, 2
    ciclo4:
        call delay100ms
        dec r23
        brne ciclo4
ret
delay500ms: ldi r24, 5
```

Captura 8.6: Código - Delay 1

```
delay500ms: ldi r24, 5
    ciclo5:
        call delay100ms
        dec r24
        brne ciclo5
ret
```

Captura 8.7: Código - Delay 2

En principio, como es descrito en las líneas de código que integran este programa, se sensa el estado del primer switch, dentro de la etiqueta "pollingswitch", se pregunta por el estado del bit 0 del puerto d, donde se conecta este switch, que fue configurado activo en bajo (0), por medio de una configuración de resistencia pull-up. Una vez preguntado si este no está activo, se salta a preguntar por el switch 2. Caso contrario, si el switch 1 está activo, vuelve a preguntar por los demás para que en el caso de que el switch fuese desactivado, se vaya a preguntar por los demás. Este principio es aplicado a los 3 switches, de modo que terminarán de hacer su secuencia una vez han sido presionados y pueden ser, de cierta forma, interrumpidos por los demás switches.

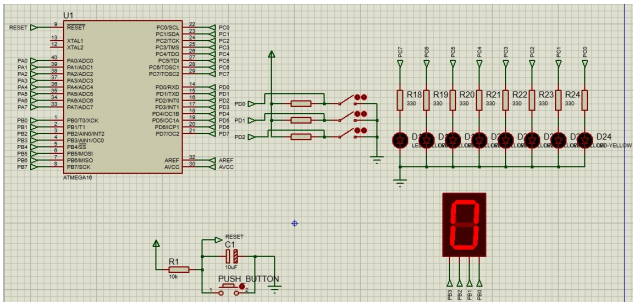
Para comprobar el funcionamiento del sistema, se utilizará Proteus en su versión 8.8, donde se carga el archivo .hex que es generado en el directorio del proyecto.

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.

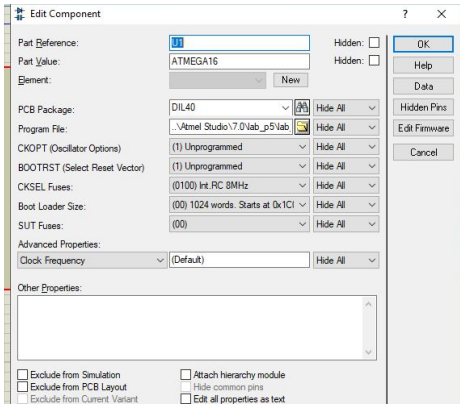
Las entradas para el sistema serán 3 switches configurados para ser activos en bajo (resistencia pull-up).

Las salidas en este caso, serán 3 puertos del ATMega 16.



Captura 9: Sistema Mínimo del ATMega16 en conexión a sus entradas y salidas.

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.

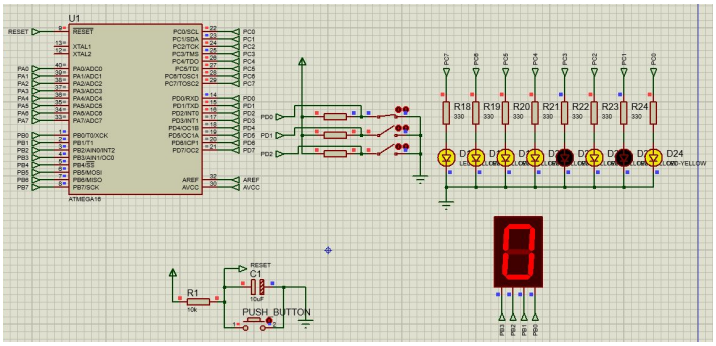


Captura 10: Carga de archivo ".hex" para la simulación de corrimiento de LEDs.

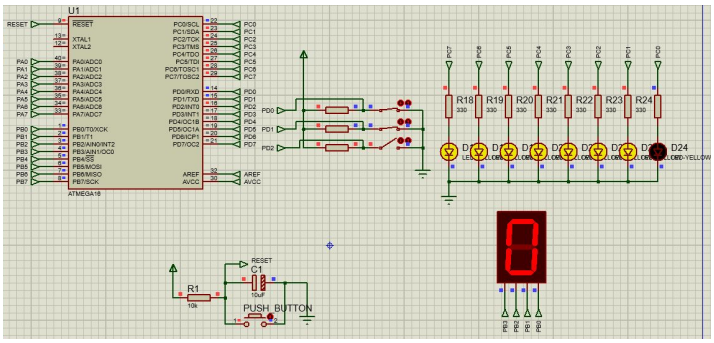
Para la selección del la frecuencia a la que trabajará el MCU, se lo hace en el apartado de "CKSEL Fuses" que prepara al MCU a trabajar de cierta forma, en este caso específico "0100", lo que significa que funcionará con su reloj interior, hecho con un arreglo de RC a 8MHz.

V. RESULTADOS

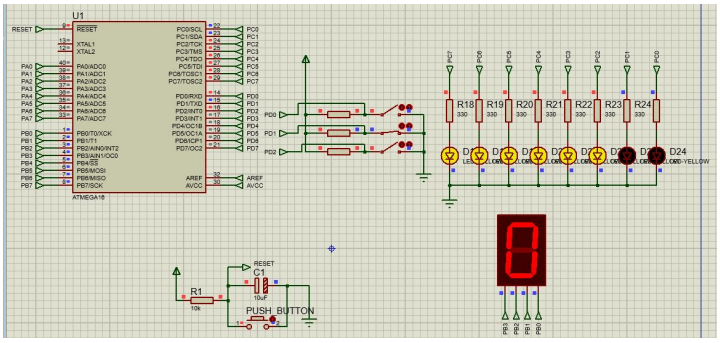
A. Switch 1 - Corrimiento Secuencia A



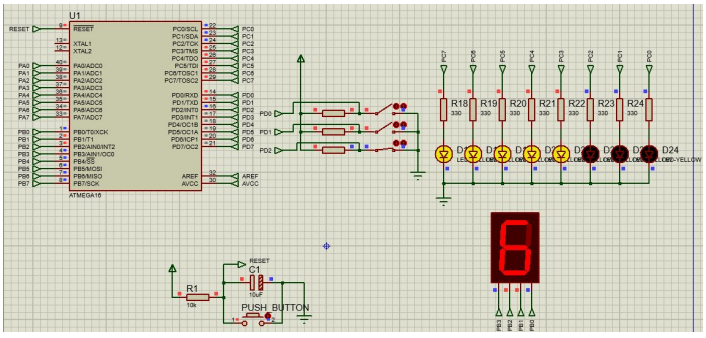
Captura 11: Ejecución Secuencia A.



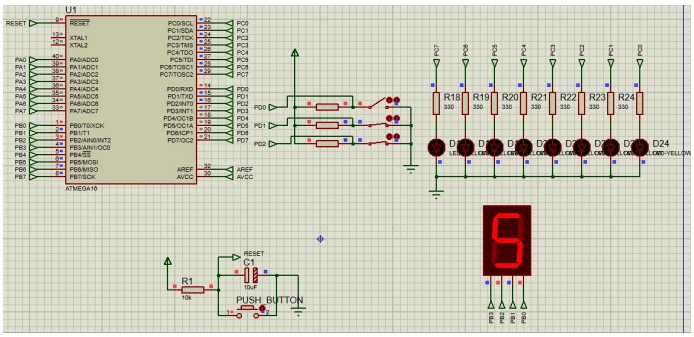
Captura 12: Detención de Secuencia A . Activación de dos Switches a la vez (1 Y 2).



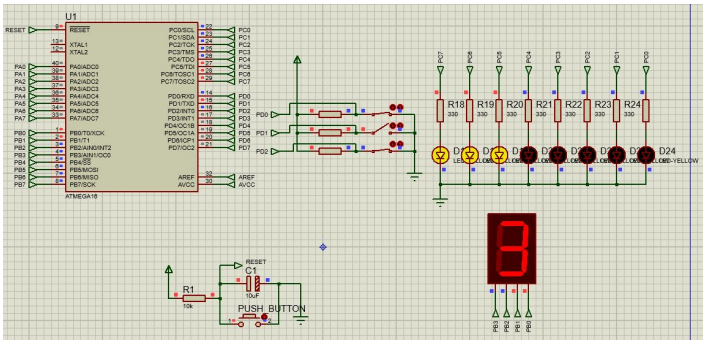
Captura 13 : Ejecución de Secuencia B.



Captura 14 : Ejecución de Secuencia C.



Captura 15: Detención de Secuencia C . Activación de dos Switches a la vez (2 Y 3).



Captura 16: Detención de Secuencia C . Activación de dos Switches a la vez (1 Y 3)

VI. Conclusiones

Andrés Islas Bravo:

Esta práctica tuvo una mayor complejidad, no en el aspecto de código sino en el aspecto de la estructuración del mismo, ya que se tuvo que tener una estructura lógica la cual permitirá dar preferencia a ciertos switches sobre otros. Por estas razones el haber realizado una estructuración del código mediante un diagrama de flujo facilitó la realización del mismo, ya que una vez decidido cómo sería la logística la implementación se realizó de manera fluida y certera. Clara prueba de las ventajas de los diagramas de flujo fue que en esta ocasión se implementaron muchos más periféricos y se tuvo un control exitoso sobre los mismos.

Juan Pablo Valverde López:

Existe libertad de diseño como en muchos lenguajes de programación, la solución no es única. Sin embargo, siempre se busca que se optimicen los recursos del sistema (AT Mega 16). En este caso se decidió que no se podrían activar dos switches a la vez, la principal restricción era que la secuencia A y B usaban el mismo puerto de salida (B) y dos o más switches se activaban, el programa se detenía y no ejecutaba nada, sensando qué switch permanece activado, luego de que dos hayan sido desactivados para poder ejecutar esa secuencia.

Alfredo Zhu Chen:

Esta práctica me permitió configurar los puertos del microcontrolador ATmega16 y también trabajar con diferentes periféricos de entrada y de salida para hacer un sistema que pueda hacer 3 diferentes secuencias. Así mismo, pude probar los retardos de software y probarlos con ATMEL Studio. La práctica fue interesante y pude reforzar y aprender conocimientos con respecto a las instrucciones del lenguaje ensamblador. Finalmente, logré probar el código escrito en el programa de simulación de Proteus para observar el funcionamiento de manera visual.

VII. Referencias

[1] ATMEL. (2010). 8-bit AVR Microcontroller with 16K Bytes In-System Programmable Flash – AtMega Atmega16L. [Archivo PDF]. Obtenido desde: http://ww1.microchip.com/downloads/en/devicedoc/doc2466.pdf