Proyecto Final
Planta Embotelladora Automatizada
TWI - PWM

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 project is mainly focused on the recapitulation and situational appliance of a broad range of qualities of microcontrollers, for this particular case an ATMega16. The microcontrollers are present in many artefacts, performing basic but fundamental tasks, and now their task is the automatization of a bottling plant. This by implementing communication protocols (I2C also named as TWI for the ATMEGA family), arithmetic decisions, ADC and LCD, all searching for the fastest and, more importantly, the most optimal performance.

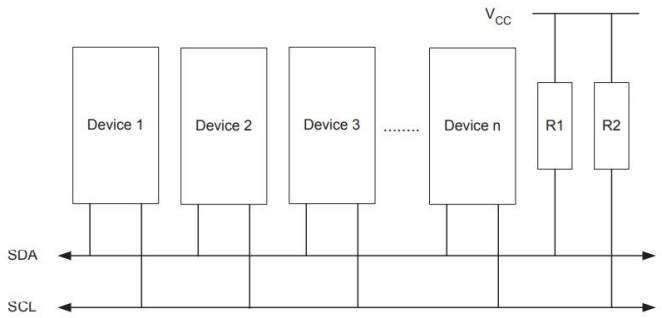
I. Introducción

En el siguiente escrito se abordarán los aspectos referentes a la automatización de una planta embotelladora. Se cubre hardware desde la perspectiva de código ensamblador y lo respectivo a PCBs, llámese parte física. El propósito sustentador de dicha automatización es evitar el estancamiento o falta de producción en cada una de las bandas consolidadoras de la planta. Se usarán lecturas ADC para brindar al usuario la oportunidad de regular la velocidad máxima de operación de su sección de trabajo, así mismo dichos valores se comparán entre todos en una unidad central con la cual existirá comunicación basada en un protocolo I2C. Por otra parte, la velocidad de labor por banda será regida por señales PWM entregadas al circuito correspondiente de un motor.

II. Marco Teórico

La comunicación serial es muy importante para poder transmitir y recibir datos entre dispositivos, es muy práctico en las aplicaciones debido a que ocupa menos espacio y permite implementar un sistema compacto. I2C (*Inter-Integrated Circuit*) o TWI*(Two-Wire Interface*) es uno de los protocolos seriales cuyo campo de aplicación se enfoca principalmente en circuitos integrados. El protocolo consiste en dos líneas, SDA(*Serial DAta*) y SCL(*Serial Clock*). Por un lado, el primero es una línea para transmitir o recibir datos de manera serial; por otro lado, el segundo es la señal de reloj para sincronizar la información e implementar un mecanismo de control en el estado de la línea de datos[1]. En ATMega 16, la velocidad de transmisión puede alcanzar hasta 400 kHz. Se conecta una resistencia de *pull-up* debido a que eléctricamente se trata de un drenador abierto en ambas líneas que cumpla con la ecuación (1), donde V_{cc} es el voltaje de alimentación y C_b es la capacitancia de una línea de BUS en pF:

$$(V_{cc} - 0.4V)/3mA \ll R \ll 1000ns/C_b \quad (1)$$



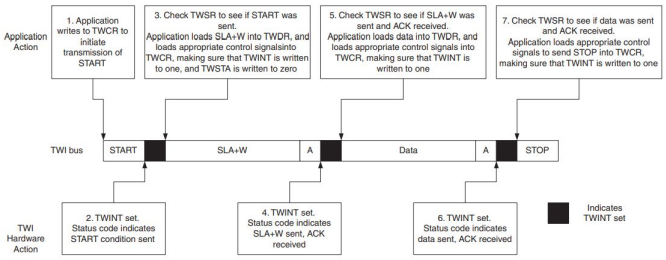
Captura 1:Diagrama de bloques para la conexión del TWI. Tomado desde: http://ww1.microchip.com/downloads/en/devicedoc/doc2466.pdf

En TWI los dispositivos pueden operar como *másters* o *slaves*, a su vez en modo transmisor o receptor. La diferencia principal es que el primero tendrá el control la línea y es quien solicitará o enviará información a los *slaves* especificados.

EL registro TWBR(*TWI Bit Register*) es el registro que permite cambiar la velocidad del reloj mediante la ecuación (2), donde TWPS(TWI Prescaler *Bits*) se encuentra en los primeros dos bits de TWSR(*TWI Status Register*), registro para conocer el estado de la línea del TWI. Así mismo, cabe mencionar que TWPS puede tomar los valores pre-escalamiento de 1,4,16 y 64 en orden para 2 bits.

$$f_{SCL} = \frac{f_{CPU}}{16+2(TWBR)\cdot 4^{TWPS}} \quad (2)$$

El registro TWCR es fundamental para controlar la operación del TWI. Se utiliza para habilitar TWI por interrupción(bit 0), habilitar TWI(bit 2), detectar con una bandera la si se trata de leer un dato cuando la línea está ocupada, mandar condición de parada la secuencia(bit 4), mandar condición de inicio(bit 5), habilitar bit de reconocimiento(*acknowledge*, bit 6) y bandera de interrupción para indicar que la línea está disponible y espera respuesta de la aplicación. Cuando se trabaja por medio de interrupción, el vector tiene que encontrarse en 0x22.



Captura 2:Secuencia de transmisión típica del TWI. Tomado desde: http://ww1.microchip.com/downloads/en/devicedoc/doc2466.pdf

El registro que almacena el dato y la dirección del *slave* con quien se quiera transmitir es el TWDR(*TWI Data Register*). Este mismo permite establecer en los últimos 7 bits la dirección del *esclavo* con un bit que indica si se quiere solicitar(bit en alto) o enviar información(bit en bajo). Para transmitir a un dispositivo específico, este último tiene que tener guardado en el registro TWAR(*TWI Address Register*) la dirección en los últimos bits y en el primer bit indicando si espera a que pueda responder una llamada general con un bit en 1. En el otro caso, puede contener el dato que se desea enviar cuando sea modo transmisor o recibir en el modo receptor.

III. Desarrollo

El presente proyecto pretende ser la primera versión de la automatización de una planta embotelladora. Emplea 3 microcontroladores ATMEGA16 que se comunican en I2C o TWI (como lo señala Atmel en su manual). El propósito de esta primera entrega es desarrollar un sistema de retroalimentación entre los 3 MCUS, con posibilidad de ampliación para diferentes bandas de producción, pues dentro de la idea donde se trabaja en este proyecto, se conciben dos líneas de producción, las cuales estarán en constante comunicación con el mcu que se denominará de ahora en adelante como *máster*, al cual entregarán su velocidad de trabajo y este les responderá con la velocidad mínima para que todas las bandas trabajan a una misma velocidad y no haya desfases en las líneas de producción, lo cual es un problema recurrente en pequeñas fábricas. La velocidad individual de las líneas de producción está dada por una resistencia variable, potenciómetro, el cual es manipulable por el operador de la línea, a su vez el *máster* recogerá aquellos valores proporcionados por las dos líneas, los comparará para regresar, como ya se lo mencionó, el más bajo. La adquisición de información del potenciómetro a los *slaves* se hará por ADC que está integrado a este tipo de MCUs. Otro punto a destacar es que una vez los *slaves* hayan recibido el valor de velocidad más bajo, estos se verán obligados a generar una modulación por ancho de pulsos (PWM) para sus respectivos motores.

En caso de que se quiera parar la producción, ya sea individual o grupal, se puede hacer con el accionar de un switch (para la detención grupal) y dos switches para la detención de línea individual. Esto hace que el sistema sea más confiable en caso de accidente o cambio de turno dentro de los operarios.

El sistema cuenta con una pantalla LCD de 16 caracteres de dos líneas, por la que se visualiza un mensaje de bienvenida a la "Planta Embotelladora".

La estructura principal de las interrupciones resultaría en:

```
;Slave1

.INCLUDE "m16def.inc"
.ORG 0
RJMP main
.ORG 0x22
RJMP twi
.ORG 0x24
RJMP stop
```

```
.ORG 0
RJMP main
.ORG 0x22
RJMP twi
.ORG 0x24
LDI R18,0b00000001
RETI
```

Captura 3: Configuración Vectores de Interrupción. Slaves. Master

Donde el vector en hexadecimal "22" corresponde a la interrupción por *two-wire-interference* o TWI, que emplea el pin 1 del puerto c (PC1) para el *system data* y el pin 0 del puerto c (PC0) para el *system clock.* En realidad estas dos líneas de puerto son las únicas necesarias para establecer comunicación entre dispositivos por I2C, pues por una línea se transmiten los datos y la otra define la velocidad a la cual se van a comunicar. Por otro lado, el próximo vector de polarización "24" corresponde a la recepción externa 2.

Dentro de lo que se denominó como "main", se encuentra la inicialización de las palabras de control para la comunicación, la pila y la rutina para el funcionamiento del display LCD (esto para el master). .

```
;Inicializar twi
;TWPS=1,  TWBR=(8MHZ/15k-16)/2/4=62
LDI R16,62  ;SCL de 100kHz, prescaler 1
OUT TWBR,R16
LDI R16,0   ;Prescaler 1
OUT TWSR,R16
```

Captura 4.1: Inicialización General dentro del main master

```
ciclo:
    CLT ;T en 0 para recepción
    LDI R25,0x40
    LDI R16,0b10100101 ;TWINT,TWSTA,TWEN,TWIE Enviar condición de START
    OUT TWCR,R16
    CLT ;T en 0 para recepción
    LDI R25,0x42
    LDI R16,0b10100101 ;TWINT,TWSTA,TWEN,TWIE Enviar condición de START
    OUT TWCR,R16
    CLR R25 ;indica la dirección general CALL en registro valor 0
    SET ;T en 1 para transmsión
    LDI R16,0b10100101 ;TWINT,TWSTA,TWEN,TWIE Enviar condición de START
    OUT TWCR,R16
    RJMP ciclo
```

Captura 4.2: Lógica inicialización main master

Para la interrupción se tiene:

```
twi:
    IN R15,SREG
    PUSH R15
    ;Esperar a que se termine de transmitir START
    wait1:
        IN R16, TWCR
        SBRS R16, TWINT
        RJMP wait1
    ;Confirmar condición de START
    IN R16, TWSR
    ANDI R16, 0b11111000    ;Mascara
    CPI R16, 0x08   ;Checar si el START se envió correctamente
    BRNE error1
    ;Enviar SLA dirección del SLAVE
    BRTC SLA_R  ;Checar si se quiere enviar o recibir
    OUT TWDR, R25       ;Dirección del Slave+W
    LDI R16,0b10000101  ;TWINT,TWEN,TWIE
    OUT TWCR, r16           ;Enviar
    RJMP wait2  ;Esperar a que se termine de enviar
    SLA_R:
        MOV R16,R25
        INC R16 ;Dirección del Slave+R
        OUT TWDR, R16
        LDI R16,0b10000101  ;TWINT,TWEN,TWIE
        OUT TWCR, r16           ;Enviar
        RJMP wait2  ;Esperar a que se termine de enviar
    ;Esperar a que se termine de transmitir SLA+W/R
```

Captura 5. Interrupción externa "tecla"

Para la detección de errores:

```
error1:
    RJMP error
    wait2:
        IN R16, TWCR
        SBRS R16, TWINT ;Esperar a queTWINT esté en 1
        RJMP wait2
    ;Confirmar SLA+W
    IN R16, TWSR
    ANDI R16, 0b11111000    ;Máscara
    BRTC confirmarSLA_R
    CPI R16, 0x20   ;Checar si se recibió SLA+W, ACK no recibido
    BREQ twiSTOP
    CPI R16, 0x18 ;Checar si se recibió SLA+W, ACK recibido
    BRNE  error
    RJMP dataByte1
    confirmarSLA_R:
        CPI R16, 0x48   ;Checar si se recibió SLA+R, ACK no recibido
        BREQ twiSTOP
        CPI R16,0x40    ;Checar si se recibió SLA+R, ACK recibido
        BRNE  error
    ;Enviar dato
```

Captura 6. Rutina detección errores master

```
dataByte1:
CPI R18,0b00000001
BRNE enviarPWM
OUT TWDR,R18
RJMP enviarSeguir
enviarPWM:
BRTC recibirByte
CP R11,R12
BRLO enviarR11
OUT TWDR,R12
RJMP enviarSeguir
enviarR11:
    OUT TWDR,R11
enviarSeguir:
//LDS R16,0x61
//OUT TWDR, R16
LDI R16,0b10000101  ;TWINT,TWEN,TWIE
OUT TWCR, R16   ;Enviar
RJMP wait3
recibirByte:
    LDI R16,0b11000101  ;TWINT,TWEA,TWEN,TWIE
    OUT TWCR, R16
;Esperar a que este disponible la línea
wait3:
IN R16, TWCR
SBRS R16, TWINT
RJMP wait3
```

Captura 7. Rutina envío dato master

```
confirmarByte:
BRTC confirmarRecepcionByte
IN R16, TWSR
ANDI R16, 0b11111000
CPI R16, 0x28   ;data byte transmitido ,ACK recibido
BRNE  error
RJMP twiStop
confirmarRecepcionByte:
    IN R16, TWSR
    ANDI R16, 0b11111000
    CPI R16, 0x50   ;data byte recibido ,ACK transmitido
    BRNE  error

    ;Liberar la línea y terminar
    LDI R16,0b11000100  ;TWINT,TWEA,TWEN
    OUT TWCR,R16

    ;Leer dato recibido
    CPI R31,2
    BREQ leerPrimerByte
    CPI R31,1
    BREQ leerSegundoByte
    leerPrimerByte:
        IN R11,TWDR
        DEC R31
        RJMP twiSTOP
    leerSegundoByte:
        IN R12,TWDR
        LDI R31,2
        RJMP twiSTOP
;Enviar STOP
```

Captura 8. Rutina confirmación datos master

```
    ;Enviar STOP
    twiStop:
    LDI R16,0b10010101  ;TWINT,TWEN,TWSTO,TWIE
    OUT TWCR, r16
    POP R15
    OUT SREG,R15
    RETI

error:
    SBI DDRC,7
    SBI PORTC,7
    POP R15
    OUT SREG,R15
    RETI
```

Captura 9. Rutina stop master

Las subrutinas “sendlcd_4bits”, “initlcd_4bits”, “retardo10ms”, “retardo500us”, “retardo40us” no serán explicadas en este informe debido a que son subrutinas utilizadas con anterioridad.

Con respecto a los *slaves,* se mantiene una estructura similar que es detallada a continuación. Por ejemplo, dentro de la inicialización, se asigna la dirección al slave como “40” en hexadecimal, este número se le asigna al TWAR.

```
;Inicializar twi
;TWPS=1,  TWBR=(8MHZ/100kHz-16)/2/4=8
LDI R16,62  ;SCL de 100kHz, prescaler 1
OUT TWBR,R16
LDI R16,0b01000101  ;TWEA,TWEN,TWIE esperar master
OUT TWCR,R16
LDI R16,0   ;prescaler de 1
OUT TWSR,R16
LDI R16,0x41    ;Dirección Slave+1(general call)
OUT TWAR,R16
```

Captura 10: Asignación de dirección e inicialización.

Comprobación estado de la comunicación, verificando el registro TWSR.

```
twi:
    IN R15,SREG
    PUSH R15
    ;Esperar a que se apague TWINT
    wait1:
    IN R16, TWCR
    SBRS R16, TWINT
    RJMP wait1
    ;Confirmar dirección
    IN R16, TWSR
    ANDI R16, 0b11111000    ;Mascara
    //OUT PORTA,R16
    CPI R16,0x60    ;Checar si fue direcición mismo+W, ACK transmitido
    BREQ recibir
    CPI R16,0xA0    ;Checar si fue direcición mismo+W
    BREQ recibir
    CPI R16,0x68    ;Checar si fue direcición mismo+W, ACK transmitido
    BREQ recibir
    CPI R16,0xA8    ;Checar si fue direcición mismo+R, ACK transmitido
    BREQ enviar
    CPI R16,0xB0    ;Checar si fue direcición mismo+R, ACK transmitido
    BREQ enviar
    CPI R16,0x70    ;Checar si fue general call+W, ACK transmitido
    BREQ recibir
    CPI R16,0x78    ;Checar si fue general call+W, ACK transmitido
    BREQ recibir

    RJMP error
```

Captura 11. Lógica detección del estado del TWSR en slave.

La recepción y transmisión era controlada por la bandera T, quien dictaba la acción a proceder del *slave.*

```
recibir:
    CLT ;T en 0 para recibir
    LDI R16,0b11000101  ;TWINT,TWEA,tWEN, twiE
    OUT TWCR,R16
    RJMP wait2

enviar:
    SET ;T en 1 para enviar
    ;Enviar dato
    OUT TWDR, R24
    LDI R16,0b10000101  ;TWINT,TWEN
    OUT TWCR, R16   ;Enviar


;Esperar a que se termine de recibir dato ACK, o a que termine de transmitir el dato
wait2:
IN R16, TWCR
SBRS R16, TWINT
RJMP wait2
```

Captura 12.Estado de la bandera T. Transmisión - Recepción en slave.

```
;Confirmar Dato
BRTS confirmarDatoTransmision
IN R16, TWSR
//OUT PORTA,R16
ANDI R16, 0b11111000    ;Mascara
CPI R16, 0x80   ;Checar si se recibió correctamente, ACK transmitido
BREQ seguirConfirmarDatoRecepcion
CPI R16,0x90
BREQ seguirConfirmarDatoRecepcion
RJMP error
seguirConfirmarDatoRecepcion:
    RJMP leerDatoRecepcion
    confirmarDatoTransmision:
        IN R16, TWSR
        ANDI R16, 0b11111000    ;Mascara
        CPI R16, 0xC8   ;Checar si se transmitió correctamente, ACK regresado
        BRNE error      ;--------------------Checar
        LDI R16,0b11000101  ;TWINT,TWEA,TWEN
        OUT TWCR,R16
        POP R15
        OUT SREG,R15
        RETI
```

Captura 13. Confirmación de datos en slave.

```
    leerDatoRecepcion:
        LDI R16,0b11000101  ;TWINT,TWEA,TWEN
        OUT TWCR,R16
        ;Leer dato
        IN R16,TWDR
        MOV R25,R16


    POP R15
    OUT SREG,R15
    RETI


error:
    SBI DDRC,7
    SBI PORTC,7
    //LDI R16,0b11000101    ;TWINT,TWEA,TWEN
    //OUT TWCR,R16
    POP R15
    OUT SREG,R15
    RETI
```

Captura 14.Lectura de datos en slave - error.

Sin embargo, como su nombre lo demuestra el *slave* es quien va a llevar la mayor carga de trabajo en lo que respecta a la programación pues es el encargado de leer el ADC y guardarlo en registros que serán enviados para la comparación dentro del *máster,* además de la generación de PWM dictada también por el *máster* para poder regular la velocidad de los motores en cada línea.

```
;PWM
LDI R16,0b01101001  ;Clear para 0C0,fast PWM, 1024 prescaler    8M/(1024*256)=30.5Hz
OUT TCCR0,R16


LDI R16,0b01100000  ;.5V de referencia, ajustamiento a la izquierda, canal 0
OUT ADMUX,R16
LDI R16,0b10000111  ;ADC enable, prescaler de 128
OUT ADCSRA,R16

SEI ;Habilitar Interrupciones
```

Captura 15. Preparación PWM en slave.

```
fin:
    SBI ADCSRA,ADSC
    pollingADC:
        SBIS ADCSRA,ADIF
        RJMP pollingADC
    SBI ADCSRA,ADIF
    //IN R16,ADCL
    IN R24,ADCH
    OUT PORTD,R24
    CPI R25,0b00000001
    BREQ errorSTOP
    OUT OCR0,R25     ;Duty Cycle
    RJMP fin
    errorSTOP:
        SBI PORTC,6
        CLR R16
        OUT TCCR0,R16
    RJMP fin
```

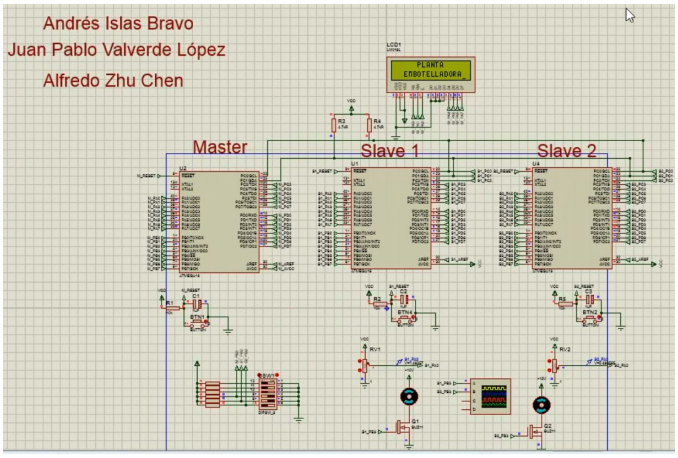
Captura 16. Lógica lectura ADC en slave.

```
stop:
    PUSH R16
    IN R15,SREG
    PUSH R15

    SBI PORTC,6
    CLR R16
    OUT TCCR0,R16

    POP R15
    OUT SREG,R15
    POP R16
    RETI
```

Captura 17. Detención de las líneas de producción.

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.

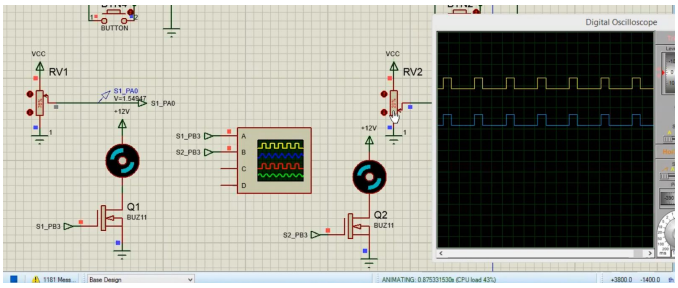
Comprobación por Proteus.



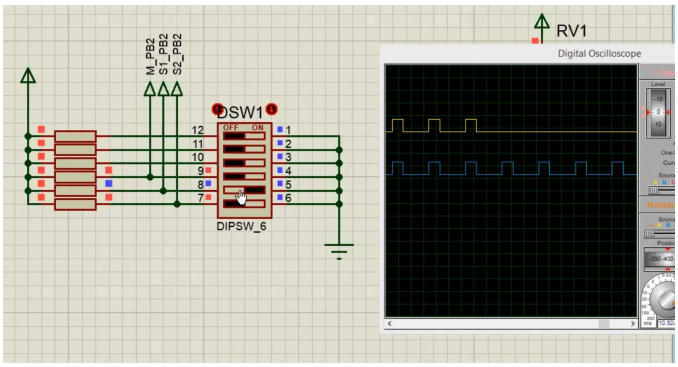
Captura 18: Sistema Planta Embotelladora.

La frecuencia a la cual trabajará el ATMega16 se mantiene, por lo que se configura "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.

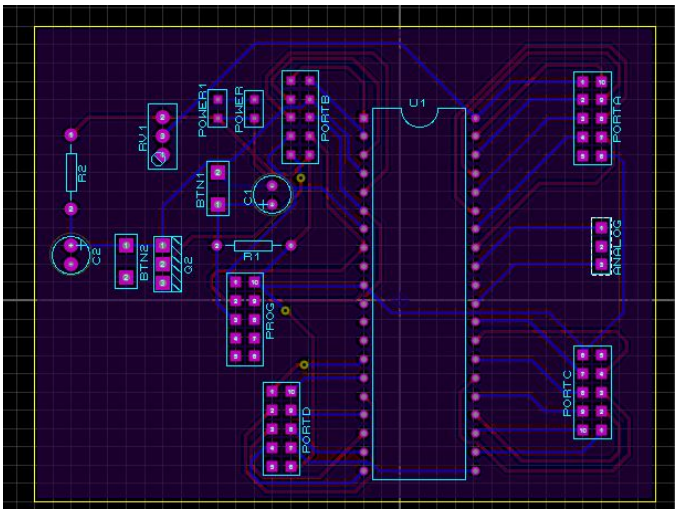
IV. Resultados



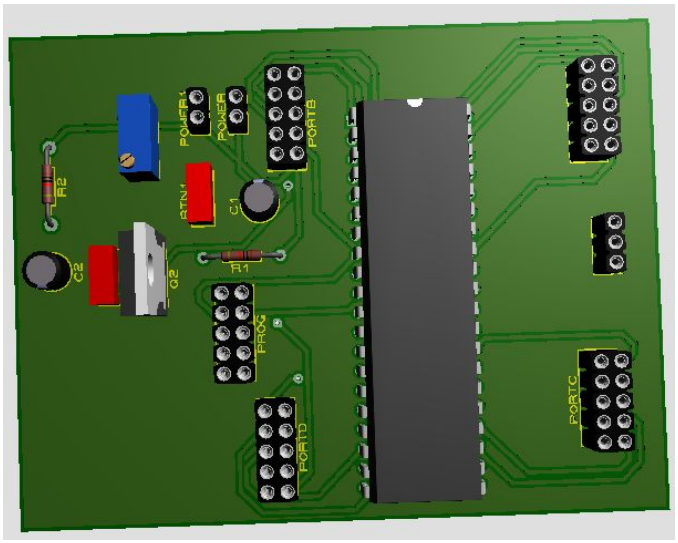
Captura 19: Retroalimentación velocidad mínima para ambas líneas de producción.



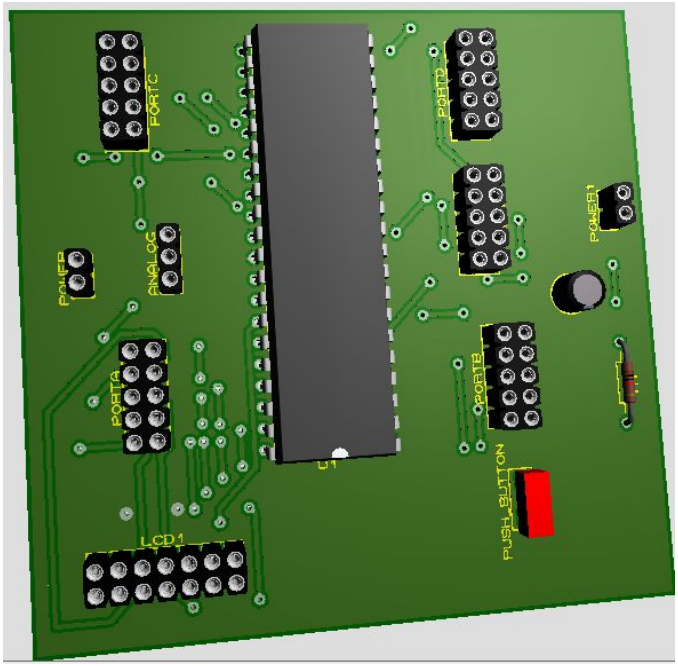
Captura 20: Detención banda individual de producción.



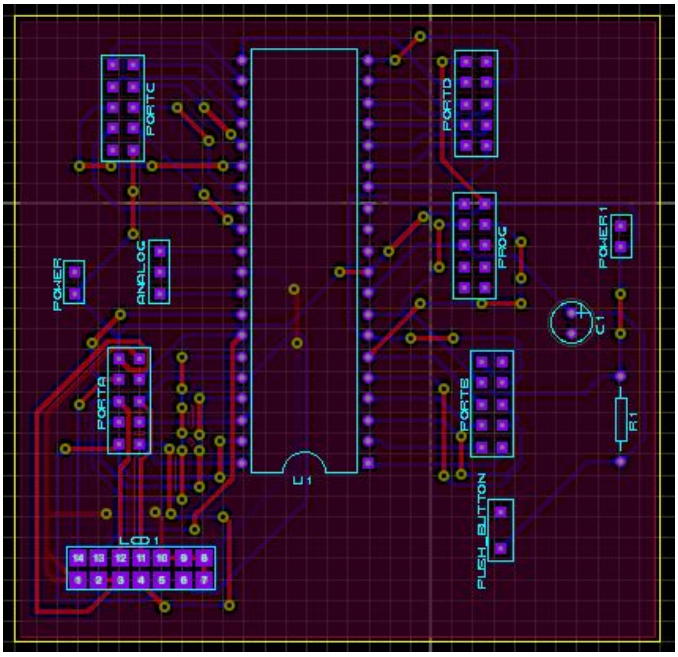
Captura 21: PCB del slave1 (misma para slave 2)



Captura 22: PCB del slave 1 -vista 3D(misma para el slave 2)



Captura 23: PCB del máster - vista 3D



Captura 23: PCB del máster - vista esquemática

V. Conclusiones

Andrés Islas Bravo:

Fue un proyecto integrador el cual permitió la recapitulación de una serie considerable de temas abordados a lo largo del curso. Se enfrentaron a diversos retos y se demandó el dominio de un protocolo de comunicación el cual fue presentando complicaciones durante su elaboración, las cuales fueron superadas tras una serie de iteraciones. Por otra parte se tuvieron limitantes a consecuencia del simulador, el cual lamentablemente no fue capaz de brindarnos una simulación en tiempo real, sin embargo representó un gran aliado ante la situación actual.

Juan Pablo Valverde López:

La implementación de un tipo de comunicación no vista en clase siempre tiene sus contratiempos y más aún cuando la documentación dentro del hardware utilizado son escasos. Si bien es cierto que la única y más poderosa referencia es el manual de usuario, descifrar e interpretar de cierta forma las tecnicidades de estas comunicación implicó un reto. Además, sumarle la adquisición de datos por medio de ADC para poder enviar datos al máster y que este retroalimente a los slaves y por último la generación de PWM. Estoy muy satisfecho con la conclusión de esta primera versión, y dada las circunstancias en simulación.

Alfredo Zhu Chen:

Este proyecto final me permitió manejar distintos conceptos vistos a lo largo del curso y aplicarlos en una misma aplicación. El uso de la comunicación serial I2C/TWI es muy interesante y creo que tiene mucha utilidad para la comunicación entre dispositivos. El objetivo del proyecto se

cumplió porque se pudo implementar la transmisión y recepción de forma serial entre distintos microcontroladores. Fue un reto estudiar un protocolo ampliamente usado en la industria y tiene muchas aplicaciones para que lo usemos. Finalmente, logré probar con mis compañeros el código escrito en el programa sobre la simulación de Proteus para observar el funcionamiento de manera visual.

VI. Referencias

[1] Moreno, A. (2004). El BUS I2C. recuperado de Bus I2C (uco.es)

[2] 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

VII. Anexos

[Código Funcionamiento]
https://drive.google.com/drive/folders/1oXEt0PBZuaEKffRTwI85FPVZL3931JIf?usp=sharing
[Video Funcionamiento]
https://drive.google.com/file/d/14uf6VHgOOV_H1O5eyekC-zS3FxPyH7FE/view?usp=sharing