

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería



Estudio comparativo de los módulos UART, PWM, I2C, SPI y ADC entre los microcontroladores PIC16F887 y ATmega328P.

Trabajo de graduación presentado por Edgar Oliverio Fajardo Monzón para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2023

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería



Estudio comparativo de los módulos UART, PWM, I2C, SPI y ADC entre los microcontroladores PIC16F887 y ATmega328P.

Trabajo de graduación presentado por Edgar Oliverio Fajardo Monzón para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

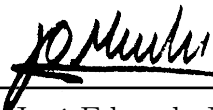
2023


Vo.Bo.:

(f) 
Ing. Kurt Kellner

Tribunal Examinador:

(f) 
Ing. Kurt Kellner

(f) 
M. Sc. José Eduardo Morales

(f) 
MAEB. Pablo Mazariegos

Fecha de aprobación: Guatemala, 3 de enero de 2023.

Prefacio

La programación de microcontroladores ha sido, desde que la conozco, mi área favorita de la Ingeniería Mecatrónica. Al haberme otorgado la oportunidad de desarrollar esta investigación, acepté sin dudar. No tengo duda alguna que fue una gran decisión, pues disfruté realizar la experimentación de la misma. Me siento orgulloso de haber aprendido un lenguaje de ensamblador distinto al que estoy habituado y poder aportar en el aprendizaje de los lectores.

Quiero agradecerle especialmente al Ingeniero Kurt Kellner, por la propuesta de investigación y la oportunidad de programar nuevamente en ensamblador. Una mención especial a mi colega y amigo Javier Schwendener, por acompañarme en este tema de investigación, complementando con los módulos faltantes.

Edgar Oliverio Fajardo Monzón
Guatemala, 20 de diciembre de 2022.

Prefacio	v
Índice general	ix
Lista de figuras	xii
Lista de cuadros	xv
Resumen	xvii
Abstract	xix
I Introducción	1
II Antecedentes	3
A Comparación PIC vs AVR	3
B Syllabus de universidades para enseñanza de programación de microcontroladores	4
C Comparación de arquitecturas de 8 bits	5
III Justificación	7
IV Objetivos	9
A Objetivo general	9
B Objetivos específicos	9
V Alcance	11
VI Marco teórico	13
A Generalidades de un microcontrolador	13
B Puertos de entrada y salida (<i>I/O Ports</i>):	13
C Oscilador:	13
D Módulo temporizador (<i>Timer</i>):	14
E Conversores de analógico a digital (ADC):	14
F Modulación por ancho de pulso (PWM):	14
G Transmisor-Receptor asíncrono universal (UART):	14

H	Interfaz periférica serial (SPI):	14
I	Circuito inter-integrado (I2C):	15
J	Microcontrolador PIC16F887	15
K	ATmega328p	15
L	Analizador lógico	16
M	Compilador	16
N	Lenguaje ensamblador	16
VII Módulo Conversor Analógico a Digital (ADC)		17
A	Módulo ADC en PIC16F887	17
1	Registros y funcionamiento	17
2	Selección del oscilador de conversión	18
3	Lectura sin interrupción	20
4	Lectura con interrupción	20
5	Lectura de múltiples entradas	22
6	Resumen de registros utilizados	23
B	Módulo ADC en ATmega328P	23
1	Registros y funcionamiento	23
2	Selección del oscilador de conversión	24
3	Lectura en modo de conversión única sin interrupción	25
4	Lectura en modo de conversión única con interrupción	26
5	Lectura en modo de corrida libre	26
6	Lectura de múltiples entradas	27
7	Resumen de registros utilizados	28
C	Comparación módulos ADC: PIC16F887 y ATmega328P	28
1	Comparación de cantidad de instrucciones y tiempo de ejecución de códigos	30
2	Comparación de cantidad de registros utilizados por módulo	32
VIII Módulo de Modulación de Ancho de Pulso (PWM)		33
A	Módulo PWM en PIC16F887	33
1	Registros y funcionamiento	33
2	Resumen de registros utilizados	35
B	Módulo PWM en ATmega328P	35
1	Registros y funcionamiento	35
2	Cálculos de PWM	37
3	Código de ejemplo, Fast PWM normal	38
4	Resumen de registros utilizados	39
C	Comparación módulos PWM: PIC16F887 y ATmega328P	39
1	Comparación de tamaño y tiempo de ejecución de códigos	40
2	Comparación de cantidad de registros utilizados por módulo	41
IX Módulo de Recepción y Transmisión Asíncrona Universal (UART)		43
A	Módulo UART en PIC16F887	43
1	Registros y funcionamiento	43
2	Configuración y cálculo de Baudrate	44
3	Envío de datos	45
4	Recepción de datos	46

5	Resumen de registros utilizados	46
B	Módulo UART en ATmega328P	46
1	Registros y funcionamiento	46
2	Envío de datos	48
3	Recepción de datos	48
4	Resumen de registros utilizados	49
C	Comparación módulos UART: PIC16F887 y ATmega328P	49
1	Comparación cantidad de instrucciones y ciclos de reloj	49
2	Comparación de cantidad de registros utilizados por módulo	53
X	Módulo de comunicación Interfaz Periférica Serial (SPI)	55
A	Módulo MSSP, configuración SPI en PIC16F887	56
1	Comunicación por estándar SPI: Maestro	57
2	Comunicación por estándar SPI: Esclavo	58
3	Resumen de registros utilizados	58
B	Módulo SPI en ATmega328P	59
1	Comunicación por estándar SPI: Maestro	60
2	Comunicación por estándar SPI: Esclavo	60
3	Resumen de registros utilizados	62
C	Comparación módulos MSSP modo SPI de PIC16F887 y Módulo SPI de ATmega328P	62
1	Respuesta de microcontroladores ante envío de datos en ráfaga	63
XI	Módulo de comunicación circuito inter-integrado (I2C)	67
A	Módulo MSSP, configuración I2C en PIC16F887	68
1	Comunicación I2C: Esclavo	68
2	Comunicación I2C: Maestro	71
3	Resumen de registros utilizados	75
B	Módulo interfaz de 2 conexiones (TWI), compatible con I2C Philips, del ATmega328P	77
1	Comunicación I2C: Maestro	77
2	Comunicación I2C: Esclavo	82
3	Resumen de registros utilizados	85
C	Comparación módulos MSSP modo I2C del PIC16F887 y Módulo TWI compatible con I2C del ATmega328P	85
1	Respuesta de microcontroladores ante envío de datos en ráfaga	86
XII	Conclusiones	89
XIII	Recomendaciones	91
XIV	Bibliografía	93
XV	Anexos	95
XVI	Glosario	97

Lista de figuras

1	Diagrama de pines del microcontrolador PIC16F887, empaquetado DIP de 40 pines.	15
2	Diagrama de pines del microcontrolador ATmega328P empaquetado DIP de 28 pines.	16
3	Comparación de tiempo de lecturas analógicas en ensamblador para PIC16F887 y ATmega328P.	28
4	Comparación de tiempo entre lecturas analógicas en ensamblador para PIC16F887 y ATmega328P.	29
5	Comparación de tiempo de lecturas analógicas en C para PIC16F887 y ATmega328P.	30
6	Comparación de tiempo entre lecturas analógicas en C para PIC16F887 y ATmega328P.	30
7	Comparación de respuesta del módulo PWM en ensamblador para PIC16F887 y ATmega328P	39
8	Modos SPI según configuración del reloj. [11]	56
9	Diagrama de flujo para comunicación por estándar SPI, rol maestro para PIC16F887.	57
10	Diagrama de flujo para comunicación por estándar SPI, rol esclavo para PIC16F887.	58
11	Modos SPI según configuración del reloj [12].	59
12	Diagrama de flujo para comunicación por estándar SPI, rol maestro para ATmega328P.	61
13	Diagrama de flujo para comunicación por estándar SPI, rol esclavo para ATmega328P.	61
14	Fragmento de señales de canales SDI, SDO, SCK y fallo, para comunicación SPI en ATmega328P.	63
15	Vista completa de señal de fallo para comunicación SPI en ATmega328P.	64
16	Fragmento de señales de canales SDI, SDO, SCK y fallo, para comunicación SPI en PIC16F887.	64
17	Vista completa de señal de fallo para comunicación SPI en PIC16F887.	65

18	Diagrama de flujo para escritura en esclavo I2C en PIC16F887	70
19	Diagrama de flujo para lectura en esclavo I2C en PIC16F887	71
20	Diagrama de flujo para escritura en Maestro I2C en PIC16F887.	74
21	Diagrama de flujo para lectura en Maestro I2C en PIC16F887.	76
22	Diagrama de flujo para escritura en maestro I2C en ATmega328P	80
23	Diagrama de flujo para lectura en maestro I2C en ATmega328P.	81
24	Diagrama de flujo para lectura en esclavo I2C en ATmega328P.	83
25	Diagrama de flujo para escritura en esclavo I2C en ATmega328P.	84
26	Fragmento de señales en error, para canales SCK, SDA y fallo, comunicación I2C en PIC16F887.	86
27	Vista completa de señal de fallo para comunicación I2C en PIC16F887.	86
28	Fragmento de señales en canales SCK, SDA y fallo para comunicación I2C en PIC16F887.	87
29	Vista completa de señal de fallo para comunicación I2C en ATmega328P.	87

Lista de cuadros

1	Canal analógico correspondiente a cada pin para registro ANSEL.	17
2	Canal analógico correspondiente a cada pin para registro ANSELH.	18
3	Tiempos de adquisición según el rango de conversión, utilizando oscilador principal.	18
4	Tiempos de adquisición según el voltaje de alimentación, utilizando oscilador dedicado del módulo ADC.	19
5	Configuraciones disponibles para selección de oscilador de ADC.	19
6	Cálculos de tiempo de adquisición según frecuencia de reloj.	19
7	Registros utilizados para el módulo ADC en PIC16F887	23
8	Selección de referencia en ATmega328P.	24
9	Selección de frecuencia de oscilador de conversión.	24
10	Cálculo de frecuencias del oscilador de conversión según prescaler para 16 MHz	25
11	Cálculo de frecuencias del oscilador de conversión según prescaler para 8 MHz	25
12	Resumen de registros utilizados en el módulo ADC en ATmega328P	28
13	Conteo de instrucciones en la configuración del ADC y periodos de reloj totales utilizados en PIC16F887, según programa propuesto, en ensamblador.	30
14	Conteo de instrucciones en la ejecución del ADC y periodos de reloj totales utilizados en PIC16F887, según programa propuesto, en ensamblador.	31
15	Conteo de instrucciones en la configuración del ADC y ciclos de instrucción totales utilizados en ATmega328P, según programa propuesto, en ensamblador.	31
16	Conteo de instrucciones en la ejecución del ADC y ciclos de instrucción totales utilizados en ATmega328P, según programa propuesto, en ensamblador.	31
17	Comparación ciclos de reloj para ADC en ensamblador.	32
18	Comparación cantidad de instrucciones necesarias para configuración y ejecución del módulo ADC en ensamblador.	32
19	Comparación de cantidad de registros utilizados por módulo	32
20	Salidas del canal 1 de PWM mejorado	33
21	Modos del canal 1 de PWM mejorado	34
22	Registros utilizados en el módulo PWM en PIC16F887	35
23	Canales y pines correspondientes	35
24	Configuración canal A de PWM 0	36
25	Configuración canal B de PWM 0	36
26	Funciones del módulo	36

27	Selección de reloj para módulo PWM	36
28	Registros utilizados en el módulo PWM en ATmega328p	39
29	Medición de error de frecuencias obtenidas de los PWM de ambos microcontroladores	40
30	Conteo de instrucciones en la configuración del PWM y periodos de reloj totales utilizados en PIC16F887, según programa propuesto en ensamblador.	40
31	Conteo de instrucciones en el cambio del ciclo de trabajo del PWM y periodos de reloj totales utilizados en PIC16F887, según programa propuesto en ensamblador.	40
32	Conteo de instrucciones en la configuración del PWM y ciclos de instrucción totales utilizados en ATmega328P, según programa propuesto en ensamblador.	40
33	Conteo de instrucciones en el cambio del ciclo de trabajo del PWM y ciclos de instrucción totales utilizados en ATmega328P, según programa propuesto en ensamblador.	41
34	Comparación ciclos de reloj para PWM en ensamblador.	41
35	Comparación de cantidad de instrucciones necesarias para configuración y ejecución del módulo PWM en ensamblador.	41
36	Comparación de cantidad de registros utilizados por módulo	42
37	Ecuaciones para configuración del baudrate en PIC16F887	44
38	Registros utilizados en el módulo UART en PIC16F887.	46
39	Modos disponibles del módulo UART	47
40	Configuraciones para modificar tamaño de carácter	47
41	configuración de Bit de paridad	47
42	Ecuaciones para cálculo del registro UBRR0	48
43	Registros utilizados en el módulo UART en PIC16F887.	49
44	Conteo de instrucciones en la configuración de la recepción de datos para UART y periodos de reloj totales utilizados en PIC16F887, según programa propuesto, en ensamblador.	50
45	Conteo de instrucciones en la recepción de datos por UART utilizando interrupciones y conteo de periodos de reloj utilizados en PIC16F887, según programa propuesto, en ensamblador.	50
46	Conteo de instrucciones en la configuración de transmisión de datos por UART y periodos de reloj totales utilizados en PIC16F887, según programa propuesto, en ensamblador.	50
47	Conteo de instrucciones en la transmisión de datos por UART y periodos de reloj totales utilizados en PIC16F887, según programa propuesto, en ensamblador.	51
48	Conteo de instrucciones en la configuración de recepción de datos en UART y ciclos de instrucción utilizados en ATmega328P, según programa propuesto, en ensamblador.	51
49	Conteo de instrucciones para recepción de datos en UART utilizando interrupciones y conteo de ciclos de instrucción utilizados en ATmega328P, según programa propuesto, en ensamblador.	51
50	Conteo de instrucciones para configuración de transmisión de datos en UART y ciclos de instrucción utilizados en ATmega328P, según programa propuesto, en ensamblador.	51

51	Conteo de instrucciones para transmisión de datos en UART y ciclos de instrucción utilizados en ATmega328P, según programa propuesto, en ensamblador.	52
52	Comparación ciclos de reloj para UART en ensamblador.	52
53	Comparación de cantidad de instrucciones necesarias para configuración y ejecución del módulo UART según programas propuestos en ensamblador. . .	52
54	Comparación de cantidad de registros utilizados por módulo	53
55	Configuración de bits SSPM [3:0] para SPI	56
56	Configuración de registros TRISx según el rol asignado para configuración SPI en PIC16F887.	57
57	Registros utilizados en el módulo MSSP: configuración SPI.	58
58	Configuración de la señal de reloj para el módulo SPI de ATmega328P	59
59	Configuración de registro DDRB según el rol asignado para configuración SPI en ATmega328P	60
60	Registros utilizados para el estándar SPI en ATmega328P	62
61	Registros utilizados para la configuración y ejecución del estándar SPI en microcontroladores PIC16F887 y ATmega328P.	62
62	Señales de indicación para comunicación I2C	67
63	Configuración de bits SSPM[3:0] para I2C.	68
64	Registros utilizados en el módulo MSSP: configuración I2C.	75
65	Códigos de estado para maestro en transmisión.	78
66	Códigos de estado para maestro en recepción.	78
67	Códigos de estado para esclavo en recepción.	82
68	Códigos de estado para esclavo en transmisión.	82
69	Registros utilizados en el módulo TWI (I2C)	85
70	Registros utilizados para la configuración y ejecución de la comunicación I2C en microcontroladores PIC16F887 y ATmega328P.	85

En la Universidad del Valle de Guatemala se imparte el curso de Programación de Microcontroladores y Electrónica Digital 2 haciendo uso del microcontrolador PIC16F887. El mismo utiliza el PICkit3 o PICkit4, los cuales presentan una serie de dificultades en cuanto a su configuración y uso. Se propone como alternativa a este microcontrolador el uso del ATmega328P, el cual es utilizado en la plataforma de desarrollo Arduino UNO.

Con el propósito de determinar la mejor opción para la enseñanza en la Universidad del Valle de Guatemala, se compara el rendimiento del microcontrolador PIC16F887 con el microcontrolador ATmega328P, esto por medio de analizadores lógicos y ciclos de reloj utilizados; las tareas realizadas son las propuestas en el curso de Programación de Microcontroladores en los laboratorios que involucran los módulos ADC, PWM, UART, I2C y SPI. Los módulos se programaron en ambos microcontroladores tanto en lenguaje ensamblador, como en C, exceptuando los módulos I2C y SPI, los cuales son programados únicamente en C.

Este estudio nos muestra mejores resultados en el ADC del ATmega328P a comparación del módulo ADC del PIC16f887, el cual tiene una resolución muy variable y lecturas más lentas así como un mayor tiempo antes de iniciar una nueva lectura. El resultado se repite en ambos lenguajes de programación, aumentando evidentemente en C. En los protocolos de comunicación evaluados, el comportamiento es equivalente y la cantidad de registros modificados depende del microcontrolador, requiriendo menos el ATmega328P.

Universidad del Valle de Guatemala (UVG) teaches courses on programming of microcontrollers, those courses uses the PIC16F887 microcontroller. This one need the PicKit3 or PicKit4 tool, both have trouble configuring and using. It is proposed as an alternative to PIC16F887, to use ATmega328P microcontroller. This is the same as the used in Arduino UNO board.

In order to determine the best option to teach in UVG, the performance is compared in both microcontrollers, using logic analyzers and machine cycles required in programs. The suggested ADC, PWM, UART, I2C and SPI programs from Programming of microcontrollers course are used to develop this investigation. Modules are programmed in both microcontrollers in Assembly language and C, except I2C and SPI, that are programmed only in C.

This investigation shows better results in the ADC Module of ATmega328P than PIC16F887, the last one shows less precision and longer resolution times as well longer times to restart a conversion. This situation continues in C language, getting longer times for both devices but even longer with PIC16F887. The performance of evaluated communication protocols are equivalent and the number of used registers depends on the microcontroller, with ATmega328P using less resgisters.

La programación de microcontroladores es un área fundamental en carreras relacionadas con la electrónica, se debe en gran medida a la utilidad de los microcontroladores para realizar tareas específicas con circuitos. Una buena forma de iniciar la enseñanza de programación de microcontroladores es con dispositivos de pocos bits. Siendo de esta forma muy común iniciar con dispositivos *Midrange* entre los cuales se pueden encontrar una amplia variedad de dispositivos de 8 bits.

La Universidad del Valle de Guatemala adoptó al PIC16F887 para impartir el curso de Programación de Microcontroladores y Electrónica Digital 2. Evaluando una mejor opción para impartir dichos cursos, se propone el ATmega328P. Tomando en cuenta el rendimiento, por medio de analizadores lógicos y ciclos de máquina utilizados en cada programa.

El estudio comparativo abarca los módulos ADC, PWM, UART, I2C y SPI; para los microcontroladores PIC16F887 y ATmega328P. Las tareas que deben completar estos módulos son basadas en las prácticas sugeridas del curso de Programación de Microcontroladores de la UVG. Para realizar una comparación justa se procura realizar el programa mínimo viable para el funcionamiento de la tarea solicitada, con el objetivo de mantener los ciclos de máquina y tiempos de respuesta en mínimos.

El estudio incluye el funcionamiento de los módulos, así como los registros involucrados con el mismo. Además, de características relevantes de cada módulo y características extras, útiles para la implementación, como interrupciones del módulo. La comparación se realiza en la parte final de cada capítulo, comparando con el analizador lógico, en los casos que lo amerita, las comparaciones y tiempos de respuesta de ambos microcontroladores; y comparando los ciclos de máquina según las instrucciones ejecutadas en los programas.

El PIC16F887 es un microcontrolador utilizado en la Universidad del Valle de Guatemala para enseñanza de arquitectura de computadoras y lenguajes de programación como Assembler y C. [1] En universidades como Gujarat Technology University, se utiliza la familia de microcontroladores PIC18 [2] y en universidades como el Massachusetts Institute of Technology (MIT), se utiliza la familia AVR, específicamente el ATmega328p. Cabe resaltar que este microcontrolador es el integrado en la plataforma Arduino UNO. [3]

A continuación se describen trabajos relacionados a la comparación de Microcontroladores de 8 bits.

A. Comparación PIC vs AVR

Limor Fried, realizó una publicación en su foro personal en el año 2004. Esta publicación compara microcontroladores de AVR contra PIC, en esta investigación evalúa puntos importantes de cada familia de microcontroladores [4].

Fried evalúa varios aspectos en su análisis. Se toma en cuenta el precio, los lenguajes de programación y compiladores para cada familia. Evalúa los entornos de desarrollo (IDE), así como los kits de desarrollo, facilidad para principiantes en programación de microcontroladores y características especiales propias de cada familia.

La publicación presenta una comparación de microcontroladores de capacidades comparables y de la misma cantidad de pines. Evalúa los PIC 12F629, 16F628 y 18F452 contra ATtiny13, ATtiny2313 y ATmega32 respectivamente. Llegando a la conclusión que los precios de estos son similares, por lo que para los microcontroladores evaluados, no existía realmente una diferencia elevada de precios. La comparación de los assembler para ambas familias concluye en la eficiencia de instrucciones para AVR, así como la mejor calidad en cuanto a compilador de AVR para programación en C.

Posteriormente, evalúa los entornos de desarrollo para los cuales en ese momento, no era superior ninguno, puesto que ambos cuentan con soporte solo para Windows.

Se contrastan los kits de desarrollo de ambas familias, para PIC se expone el Pickit1 con un precio de 44\$ en el año 2004 y para AVR se presenta el STK500 con un precio de 80\$, concluyendo como mejor opción el Pickit1 por el precio. Finalmente se analiza cual de las 2 familias es más viable para un principiante, con los 2 exponentes respectivos, siendo los mismos Arduino para AVR y BASIC Stamp para PIC, debido a la facilidad que presenta Arduino, introduciendo C con macros para facilitar el aprendizaje, se le determina como superior en este aspecto.

B. Syllabus de universidades para enseñanza de programación de microcontroladores

Para la enseñanza de programación de microcontroladores en otras universidades del mundo se utilizan microcontroladores de la familia PIC como de la familia AVR, de la misma forma que existen universidades más centradas en la enseñanza de la arquitectura para un mejor entendimiento, utilizando otras plataformas de desarrollo, por lo que se presentan las siguientes universidades con sus respectivos cursos.

Gujarat Technological University, proporciona el curso de *PIC Microcontroller and embed systems* [2]. En el que se enseña a programar en PIC para procesamiento de señales y programación de sensores, motores, relés, entre otros. En este curso se enseña el desarrollo de programas en C para el microcontrolador PIC18, abarcan también la arquitectura RISC para el mismo.

Massachusetts Institute of Technology utiliza el ATmega328p en la placa de microcontrolador Arduino Uno, para proporcionar el curso de *PERFORMANCE ENGINEERING OF SOFTWARE SYSTEMS* [3], se enseña assembler para AVR así como arquitectura del microcontrolador. El curso se orienta a aprender como mejorar el rendimiento de los programas por medio del conocimiento del conocimiento profundo de la plataforma.

La Texas Tech University (TTU), imparte el curso de *Microcontrollers*[5], utilizando el microcontrolador MSP430 de Texas Instruments, en el mismo se enseña Assembler y C. Se introducen los conceptos de subrutinas, interrupciones y funciones del microcontrolador. Además, en esta Universidad se enseña la arquitectura RISC en el curso *Microprocessor Architecture* [6], en el que se diseña un procesador sencillo basado en la arquitectura RISC (arquitectura empleada en el ATmega328p).

University of Alabama (UA), se imparte el curso *Microcomputers* [7], utilizando el microcontrolador PIC24, para el cual se enseña la arquitectura interna del microcontrolador, direccionamiento de memoria y modos de direccionamiento. Impartiendo assembler y C para el mismo.

En la Universidad del Valle de Guatemala se imparte el curso de Programación de Microcontroladores [1] utilizando el microcontrolador PIC16F887. El curso instruye arquitecturas de microcontroladores, además de fundamentos de assembler para PIC, en el curso se exponen las diferentes funciones del microcontrolador y la implementación de las mismas. El mismo microcontrolador se utiliza en Electrónica digital [2][8] para programar en C así como la TIVA C TM4C123G de Texas Instruments.

C. Comparación de arquitecturas de 8 bits

En el año 1997, Microchip realizó una comparación de microcontroladores de 8 bits contra los PIC16C5X/XX [9]. En la investigación se compararon rendimiento y tiempo de ejecución de la misma tarea. Los microcontroladores comparados fueron: SGS-Thomson ST62 8 MHz, Motorola MC68HC05 4.2 MHz, Intel 8051 20 MHz, Zilog Z86CXX 12 MHz, National COP800 20 MHz

Se realizó una comparación del tiempo de ejecución para realizar la codificación de un byte, *Loop control*, comprobación de bit y direccionamiento, así como introducción de un byte en un registro y configuración del reloj, configuración de un timer. En la investigación se obtiene el promedio de tamaño de código por microcontrolador obteniendo 1.29 para el COP 800, 2.10 para el ST62, 2.24 para el MC68HC05, 1.51 para Z86CXX, 1.547 para el 8051 y finalmente 1.00 para el PIC16C5X/XX. Para los tiempos de ejecución relativos se obtienen los siguientes resultados, 0.108 para el COP 800, 0.0455 para el ST62, 0.136 para el MC68HC05, 0.212 para Z86CXX, 0.30 para el 8051 y finalmente 1.00 para el PIC16C5X/XX debido a que es el punto de comparación. Concluyendo con que el menor tamaño de código es el PIC16C5X/XX y de la misma forma en tiempos de ejecución relativos, siendo el PIC16C5X/XX el más rápido y el de mayor tiempo de ejecución es el ST62.

Justificación

La programación y conocimiento de arquitecturas de microcontroladores es un aspecto fundamental en múltiples ramas de la ingeniería. Mejorar el rendimiento de los programas requiere conocer los microcontroladores y manejar sus respectivos lenguajes ensambladores. Es la razón por la cual muchas universidades en el mundo enseñan estos aspectos en sus cursos de microcontroladores.

La Universidad del Valle de Guatemala emplea el microcontrolador PIC16F887 en cursos como Programación de Microcontroladores y Electrónica Digital 2, además de una introducción a lenguajes de ensamblador en Electrónica Digital 1. La programación de estos dispositivos ha presentado problemas, por ejemplo, de programación con el PICKit 3. Adicionalmente el costo es más elevado que otros microcontroladores y no está tan ampliamente disponible.

El objetivo de este trabajo de graduación es evaluar otro candidato disponible localmente. Este candidato es el ATmega328p utilizado en las plataformas de desarrollo Arduino UNO, el cual presenta una mayor disponibilidad en nuestra región, además de un costo inferior.

Por lo mencionado con anterioridad, se pretende realizar una comparación en términos generales de ambos microcontroladores para determinar la viabilidad de la implementación del mismo en los cursos mencionados, en la Universidad del Valle de Guatemala.

A. Objetivo general

Realizar un estudio comparativo entre los microcontroladores PIC16F887 y ATmega328P, de los módulos ADC, PWM, I2C, SPI y UART, y evaluar los parámetros necesarios para determinar la mejor opción para la enseñanza de microcontroladores en la Universidad del Valle de Guatemala.

B. Objetivos específicos

- Comparar el rendimiento del microcontrolador PIC16F887 con el microcontrolador Atmega328p por medio de analizadores lógicos, evaluando los módulos: ADC, PWM, I2C, SPI y UART.
- Realizar los laboratorios y proyectos propuestos en el curso de Programación de Microcontroladores de la Universidad del Valle de Guatemala, relacionados a los módulos a evaluar, en ambos microcontroladores.

Este estudio abarca la comparación y funcionamiento externo y de configuración de los módulos ADC, PWM, UART, I2C y SPI de los microcontroladores ATmega328P y PIC16F887, programados mediante la herramienta PicKit4. Procurando obtener los programas óptimos para cada uno, más no es el objetivo principal hallar estos programas mínimos viables. El estudio utiliza el compilador XC8 de Microchip para ambos microcontroladores utilizando tanto el assembler de PIC (PIC-AS) como el assembler de AVR (AVRASM2). Para compilación en C es utilizado el compilador base MPLAB XC8 C.

El estudio analiza únicamente la programación y rendimiento de los microcontroladores de forma externa. No se pretende indagar en la arquitectura de los mismos ni se abarca el comportamiento de los microcontroladores expuestos a diferentes condiciones ambientales, esto incluye temperaturas, voltajes y campos eléctricos o magnéticos.

A. Generalidades de un microcontrolador

Un microcontrolador es un circuito integrado programable. [10] Los mismos son utilizados para la fabricación de sistemas embebidos. Estos cuentan generalmente con una memoria flash, RAM, ALU, EEPROM, oscilador (algunos modelos antiguos de microcontroladores, no incluyen oscilador y se debe colocar un oscilador externo), entradas y salidas; las funciones incluidas en el microcontrolador, dependen del modelo, como lo podrían ser módulos de Timer, Conversores analógico a digital (ADC), módulos de PWM, así como módulos de comunicación bajo diferentes protocolos, como UART, I2C o SPI [11].

B. Puertos de entrada y salida (*I/O Ports*):

Estos son todas las entradas y salidas con capacidad de recibir o emitir un dato digital, es decir un voltaje HIGH o LOW, entre el rango definido. [10] Los microcontroladores manejan estos datos de forma binaria siendo clasificados por el microcontrolador dependiendo de los rangos de voltajes establecidos.

C. Oscilador:

Módulo fundamental para el funcionamiento del microcontrolador, este determina la frecuencia de operación del mismo. En algunos casos, este es programable para determinar diferentes frecuencias de operación [10]. Las formas de programar el mismo dependen según el microcontrolador. En otras ocasiones este módulo es remplazado por osciladores externos conocidos también como cristales.

D. Módulo temporizador (*Timer*):

Este módulo depende del fabricante, sin embargo, por lo general estos funcionan como contadores de aumento, para el cual al alcanzar un número determinado activan interrupciones en el programa. Pueden tener funciones extras como la captura y comparación de datos en tiempo, así como pre-escalados [11]. Los mismos pueden llegar a ser utilizados en otros módulos o activar interrupciones para realizar una tarea de forma periódica.

E. Conversores de analógico a digital (ADC):

Estos módulos permiten la conversión de una señal de entrada analógica a una salida binaria. La cantidad de bits utilizada para la representación de dicha señal dependen de la resolución de conversión utilizada por el módulo [12]. La comparación de voltaje se realiza generalmente al voltaje de operación del microcontrolador, sin embargo, existen algunos módulos con comparación ajustable para otros voltajes [11].

F. Modulación por ancho de pulso (PWM):

Este módulo modifica el ciclo de trabajo de un pulso periódico, con el objetivo de controlar la cantidad de energía que se envía, o bien, transmitir información [13]. Estos módulos generalmente tienen pines vinculados a estos módulos y suelen estar ligados a los módulos de *timer* así como una forma de control analógico controlado digitalmente.

G. Transmisor-Receptor asíncrono universal (UART):

Es un protocolo de comunicación para el intercambio de datos entre 2 dispositivos, este protocolo utiliza 2 canales que son para recibir y transmitir [14]. Esta comunicación puede ser Simplex, Semiduplex o duplex. Siendo unidireccional, bidireccional por turnos o bidireccional simultánea [11].

H. Interfaz periférica serial (SPI):

Es un protocolo de comunicación que utiliza como máximo 4 canales: CS, MISO, MOSI y SCK. Estos son para seleccionar el chip, para recepción del maestro, para la emisión del maestro y para control del reloj respectivamente [15].

I. Circuito inter-integrado (I2C):

Es un protocolo de comunicación serial, este protocolo permite la comunicación de 2 dispositivos digitales por medio del uso de 2 canales, los cuales constan de data y reloj (SDA y SCL)[15].

J. Microcontrolador PIC16F887

Es un microcontrolador de 8 bits, de la familia PIC, desarrollado por Microchip, con 40/44 pines dependiendo del empaquetado del mismo, basado en Flash y posee una arquitectura Harvard modificada. El PIC16F887 tiene una longitud de palabra de 14 bits. Este microcontrolador posee los siguientes módulos: Oscilador, 3 módulos de timer, de los cuales 2 son de 8 bits y 1 de 16 bits; cuenta con un módulo comparador, un módulo ADC de 13 canales, 2 canales de PWM, módulo de comunicación UART, un módulo MSSP que incluye I2C y SPI [11].

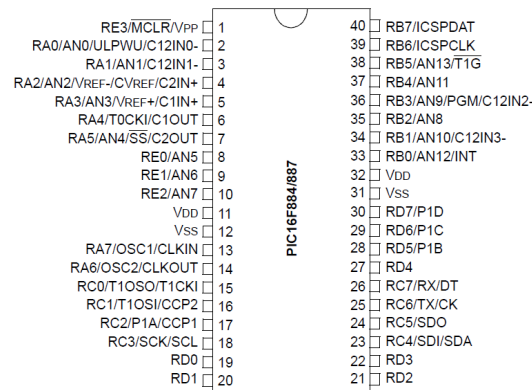


Figura 1: Diagrama de pines del microcontrolador PIC16F887, empaquetado DIP de 40 pines.

K. ATmega328p

Es un microcontrolador de 8 bits basado en AVR RISC, desarrollado por Atmel, actualmente propiedad de Microchip. Cuenta con 28/32 pines dependiendo del empaquetado, tiene una longitud de palabra de 16 bits. Este microcontrolador posee los siguientes módulos: 6 canales PWM, 3 módulos de timer, de los cuales 2 son de 8 bits y uno de 16 bits, 6 canales ADC con resolución máxima de 10 bits, un módulo comparador, módulo de comunicación UART, un módulo de comunicación SPI y un módulo de programación I2C [12].

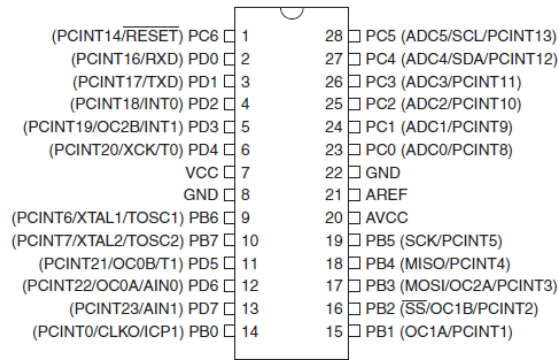


Figura 2: Diagrama de pines del microcontrolador ATmega328P empaquetado DIP de 28 pines.

L. Analizador lógico

Es un instrumento utilizado para capturar datos en un sistema digital con el objetivo de depurar errores en circuitos y verificar el funcionamiento de los mismos monitorizando el voltaje en los nodos conectados. Permite visualizar el estado de diferentes señales digitales simultáneamente. Esta herramienta permite verificar el tiempos de subida o bajada, estados lógicos y cambios de estado [16].

M. Compilador

La compilación se refiere a la conversión de un algoritmo expresado en un lenguaje humano a un algoritmo expresado en lenguaje de máquinas, por lo que el compilador traduce el código en un lenguaje hacia lenguaje de máquina previo a su implementación [17].

N. Lenguaje ensamblador

Es un lenguaje de bajo nivel, el cual es aún legible por un humano, este tiene la intención de comunicarse de forma casi directa al hardware del microcontrolador, generalmente traducido a lenguaje de máquina con el uso de un compilador. Este utiliza instrucciones básicas que se ejecutan en cierta cantidad de ciclos de máquina dependiendo de la instrucción y el dispositivo en el que se ejecuta. El lenguaje de ensamblador que se utiliza puede variar entre cada dispositivo dependiendo de su fabricante [18].

Módulo Conversor Analógico a Digital (ADC)

Tanto el PIC16F887, como el ATmega328P; cuentan con un módulo ADC que permite la conversión de una señal analógica a una señal digital. Se comparan ambos módulos en función del tiempo de conversión con las mismas configuraciones y la cantidad de registros que se modifican. Se realiza el programa en ensamblador y C para ambos microcontroladores. El logic analyzer se configura para que lea los datos a 125 MS/s, por 500 ms. A continuación se explica el módulo ADC de ambos microcontroladores.

A. Módulo ADC en PIC16F887

1. Registros y funcionamiento

El módulo posee un multiplexor (MUX) de 13 entradas analógicas. La selección de la entrada deseada, se realiza en el registro A/D CONTROL REGISTER 0 (ADCON0), en los bits [3:0]. Se puede seleccionar los pines con entrada analógica del Puerto A [5], [3:0]; del puerto E [2:0] y del puerto B [5:0]. Para los cuales, el pin o los pines a utilizar tienen que estar establecidos como entradas, utilizando el registro TRISx, correspondiente al puerto que se va a utilizar. Es necesario verificar el canal analógico del pin a utilizar, tal como se presenta en el Cuadro 1 y 2, modificando los registros ANSEL y ANSELH¹.

PIN	A0	A1	A2	A3	A5	E0	E1	E2
Canal analógico	ANS0	ANS1	ANS2	ANS3	ANS4	ANS5	ANS6	ANS7

Cuadro 1: Canal analógico correspondiente a cada pin para registro ANSEL.

¹Importante notar que los canales analógicos del registro ANSELH, no se encuentran en el mismo orden de los pines.

PIN	B2	B3	B1	B4	B0	B5
Canal analógico	ANS8	ANS9	ANS10	ANS11	ANS12	ANS13

Cuadro 2: Canal analógico correspondiente a cada pin para registro ANSELH.

Al momento de activar un bit de ANSEL o ANSELH, se activa la resistencia pull-up interna de forma automática para el pin seleccionado. Además de activar el canal analógico correspondiente.

El resultado de la conversión tiene una resolución de 10 bits que se almacenan en los registros ADRESH y ADRESL. La "justificación" del resultado puede ser hacia la izquierda o hacia la derecha, esto significa que el bit más significativo (msb), se encuentra en el msb del registro ADRESH para el caso de justificación izquierda. En caso contrario, es decir en caso de justificación derecha, el bit menos significativo (lsb), se encuentra en lsb del registro ADRESL. Para seleccionar uno de los dos casos de justificación, se debe modificar el bit ADFM del registro ADCON1, es decir el bit 7 de dicho registro. Por defecto, el microcontrolador selecciona la justificación izquierda, con el valor 0 en el bit ADFM. El registro ADCON1, puede modificar a su vez, los voltajes de referencia para la comparación del ADC, mediante los registros VCFG1 y VCFG0, conectando las referencias necesarias a los pines A2 y A3. La modificación de referencias son útiles para sensores que requieren niveles de voltaje diferentes a 5v.

Para iniciar el módulo se puede utilizar el bit ADON del registro ADCON0, este registro se puede activar en cualquier momento de la configuración de estos registros. El bit $\overline{GO/DONE}$, en cambio, se activa únicamente al momento de iniciar la conversión, este se activa posteriormente a finalizar la configuración.

2. Selección del oscilador de conversión

El PIC16F887, tiene un requisito de tiempo de adquisición de datos (T_{AD}), según la hoja de datos del PIC, este tiempo varía en función de los voltajes de referencia, contando con los tiempos presentados en el Cuadro 3. Estos tiempos de adquisición no son válidos para el oscilador dedicado del módulo ADC, para el cual, se utilizan los (T_{AD}) que se presentan en el Cuadro 4. Para el oscilador dedicado los valores varían en función del voltaje de alimentación (V_{DD}), del PIC².

V_{Ref}	Min	Max
$\geq 3.0V$	1.6 μS	9 μS
Rango entero	3 μS	9 μS

Cuadro 3: Tiempos de adquisición según el rango de conversión, utilizando oscilador principal.

²Este oscilador depende del voltaje de alimentación del PIC, el mismo puede variar entre 160KHz y 500KHz

V_{DD}	Min	Típico	Max
2.5V	3 μ S	6 μ S	9 μ S
5V	1.6 μ S	4 μ S	6 μ S

Cuadro 4: Tiempos de adquisición según el voltaje de alimentación, utilizando oscilador dedicado del módulo ADC.

Una conversión requiere 11 T_{AD} , por lo que se tiene que cumplir los requisitos presentados en los cuadros 3 y 4.

El microcontrolador cuenta con 4 configuraciones, entre las cuales escoger la frecuencia del oscilador para la conversión. Las configuraciones se presentan en el Cuadro 5, estas se seleccionan con los bits ADCS [1:0], del registro ADCON0. Se puede obtener el tiempo de adquisición a partir de la frecuencia del oscilador seleccionado y aplicando el prescaler a dicha frecuencia, el periodo del resultado presenta el tiempo de adquisición. Este mismo se compara contra los límites de T_{AD} .

Selección de oscilador	ADCS
$F_{OSC}/2$	00
$F_{OSC}/8$	01
$F_{OSC}/32$	10
Oscilador dedicado	11

Cuadro 5: Configuraciones disponibles para selección de oscilador de ADC.

F_{OSC}	Prescaler	F_{Res}	T (s)	T (μ s)	Válido
20MHz	$F_{OSC}/2$	10 MHz	1×10^{-7}	0.1	No
	$F_{OSC}/8$	2.5 MHz	4×10^{-7}	0.4	No
	$F_{OSC}/32$	625 KHz	1.6×10^{-6}	1.6	Sí
16MHz	$F_{OSC}/2$	8 MHz	1.25×10^{-7}	0.125	No
	$F_{OSC}/8$	2 MHz	5×10^{-7}	0.5	No
	$F_{OSC}/32$	500 KHz	2×10^{-6}	2	Sí
8MHz	$F_{OSC}/2$	4 MHz	2.5×10^{-7}	0.25	No
	$F_{OSC}/8$	1 MHz	1×10^{-6}	1	No
	$F_{OSC}/32$	250 KHz	4×10^{-6}	4	Sí
4MHz	$F_{OSC}/2$	2 MHz	5×10^{-7}	0.5	No
	$F_{OSC}/8$	500 KHz	2×10^{-7}	2	Sí
	$F_{OSC}/32$	125 KHz	8×10^{-6}	8	Sí
1MHz	$F_{OSC}/2$	500 KHz	2×10^{-6}	2	Sí
	$F_{OSC}/8$	125 KHz	8×10^{-6}	8	Sí
	$F_{OSC}/32$	31250Hz	3.2×10^{-5}	32	No

Cuadro 6: Cálculos de tiempo de adquisición según frecuencia de reloj.

3. Lectura sin interrupción

Se debe completar la configuración del módulo modificando los registros ADCON0 y ADCON1, posterior a este proceso, se puede iniciar la conversión AD con el bit GO/\overline{DONE} del registro ADCON0, se debe monitorizar que la conversión se complete, por lo que se debe esperar que se cumpla alguna de las dos siguientes condiciones: El bit GO se limpia o el bit ADIF del registro PIR1 se activa. El resultado se mantiene en los registros ADRESH y ADRESL.

```
CONF:
BANKSEL ANSEL
MOVLW 0b00010000 ;Canal analógico 12 activo
MOVWF ANSELH
BANKSEL TRISB
MOVLW 0b00000001 ;Seleccionar el PBO como entrada
MOVWF TRISB
BANKSEL PORTA
MOVLW 0b01110001 ;Selecciona el oscilador Fosc/8 [7:6], canal 12 [5:2]
MOVWF ADCON0 ;y se activa el módulo ADC [0]
BSF ADCON0, 1 ;Inicia la conversión

READ:
BTFSS PIR1, 6 ;Se verifica el estado del bit ADIF
GOTO $-1
MOVF ADRESH, W ;Se lee el registro ADRESH
MOVWF adcValue ;Se guarda en una variable
BCF PIR1, 6 ;Se limpia la bandera de conversión AD
BSF ADCON0, 1 ;Inicia la conversión
GOTO READ
```

La selección del oscilador se realiza tomando en cuenta la frecuencia del oscilador interno utilizado en el programa, por defecto el PIC16F887 utiliza el reloj interno de 4MHz, por lo que las configuraciones disponibles según los cálculos presentados en el Cuadro 6. Por lo que se selecciona la frecuencia de conversión más rápida.

4. Lectura con interrupción

Cuando se cuenta con interrupción, no es necesario monitorizar la conversión. Para activar las interrupciones es necesario activar los bits GIE y PIE del registro INTCON, en el caso específico de la interrupción para el ADC, se debe activar el bit ADIE del registro PIE1. Además de la necesidad de guardar los valores actuales al momento de entrar al vector de interrupción. La bandera ADIF se debe de limpiar al finalizar las operaciones de la interrupción y antes de la instrucción RETFIE. El vector de interrupción es el 0x0004, por lo que se debe colocar las instrucciones a ejecutar a partir de este espacio de memoria, cuando se tiene activa más de una interrupción es necesario verificar la interrupción.


```

PSECT intVect, class=CODE, delta=2
ORG 0x0004
SAVE:
MOVWF    w_temp      ;Guardamos W en una variable
SWAPF    STATUS, W
MOVWF    s_temp      ;Guardamos STATUS

INT:
BTFSS    PIR1, 6     ;Se comprueba si la interrupción es de ADC (ADIF)
GOTO     LOAD
MOVF     ADRESH, W   ;Se lee el resultado de la conversión
MOVWF    adcValue
BCF      PIR1, 6     ;Se limpia la bandera ADIF
BSF      ADCON0, 1   ;Se inicia la conversión nuevamente

LOAD:
SWAPF    s_temp, W
MOVWF    STATUS      ;Se regresa STATUS
SWAPF    w_temp, F
SWAPF    w_temp, W   ;Se regresa W
RETFIE

```

Como se demuestra en el código, se guarda STATUS y de W, además de regresarlo al finalizar. Se verifica el estado de la bandera ADIF y posterior a comprobarlo, guarda el resultado en una variable. Por último, limpia el estado de la bandera de interrupción para poder repetir el programa.

```

CONF:
BANKSEL  ANSEL
MOVLW   0b00010000 ;Canal analógico 12 activo
MOVWF   ANSELH
BANKSEL  TRISB
MOVLW   0b00000001 ;Selecciona el puerto PB0 como entrada
MOVWF   TRISB
MOVLW   0b11000000 ;Se activan GIE Y PIE
MOVWF   INTCON
MOVLW   0b01000000 ;Se activan las interrupciones por ADC (ADIE)
MOVWF   PIE1
BANKSEL  PORTA
MOVLW   0b01110001 ;Selecciona el oscilador Fosc/8 [7:6], canal 12 [5:2]
MOVWF   ADCON0     ;y se activa el módulo ADC [0]
BSF     ADCON0, 1   ;Inicia la conversión

```

5. Lectura de múltiples entradas

Para realizar lecturas con múltiples entradas, se debe cambiar el canal en el registro ADCON0. Esto se debe realizar posterior a finalizar la conversión. Se puede realizar sin importar si se realizan interrupciones. Para asegurar el tiempo necesario de descarga del capacitor interno del módulo, se recomienda realizar el cambio de canal posterior a leer el resultado de la conversión.

```
INT:
BTFSS  PIR1, 6      ;Se comprueba si la interrupción es de ADC (ADIF)
GOTO   LOAD
MOVLW  01110001
SUBWF  ADCON0, W
BTFSS  STATUS, 2   ;Se verifica el canal actual
GOTO   CH10
MOVF   ADRESH, W   ;Se lee el resultado de la conversión
MOVWF  adcValue1   ;Se guarda en variable de lectura 1
GOTO   SWITCH

CH10:
MOVF   ADRESH, W   ;Se lee el resultado de la conversión
MOVWF  adcValue2   ;Se guarda en variable de lectura 2

SWITCH:
MOVLW  00011000
XORWF  ADCON0      ;alterna de canal, sin modificar los demás valores
BCF    PIR1, 6     ;Se limpia la bandera ADIF
BSF    ADCON0, 1   ;Se inicia la conversión nuevamente
```

Se recomienda en Ensamblador, si se utilizan solo 2 canales, calcular un *Exclusive Or* (XOR). Pues esta operación permite alternar entre 2 canales con 2 instrucciones, sin modificar la configuración establecida previamente, además, que la misma máscara utilizada para cambiarlo de canal, sirve para revertir la operación. En este ejemplo, el cambio de canal se realiza en la sección de código "SWITCH", para modificar el canal utilizado.

```
CONF:
BANKSEL ANSEL
MOVLW  0b00010100 ;Seleccionamos los canales 12 y 10
MOVWF  ANSELH
BANKSEL TRISB
MOVLW  0b00000011 ;Activamos las entradas de los pines PB0 Y PB1
MOVWF  TRISB
BANKSEL PORTA
```

```

MOVLW    0b01110001
MOVWF    ADCON0
BSF      ADCON0, 1    ;Iniciamos la conversión

```

En esta ocasión, al utilizar 2 canales, se deben activar ambos en el registro de ANSELH correspondientes a sus canales; así como los pines utilizados como entradas analógicas.

6. Resumen de registros utilizados

Registro	Usos
ANSEL	Estos activan los puertos analógicos de los 14 disponibles.
ANSELH	
ADCON0	Este registro permite la selección del canal a utilizar, la señal de reloj que utiliza el módulo y activar el módulo, así como iniciar una conversión.
ADCON1	El registro permite justificar el resultado a la derecha o izquierda y establecer las referencias a utilizar.

Cuadro 7: Registros utilizados para el módulo ADC en PIC16F887

B. Módulo ADC en ATmega328P

1. Registros y funcionamiento

El módulo posee un MUX de 6 canales en el caso del empaquetado DIP, los cuales se pueden seleccionar mediante los bits MUX[3:0] del registro ADMUX. Se pueden seleccionar los pines con entrada analógica del Puerto C [5:0]. Para los cuales se debe configurar los pines a utilizar como entradas, con el registro DDRC, limpiando el bit correspondiente al pin; y activar las resistencias Pull-up, con el registro PORTC activando el bit correspondiente al pin a utilizar³.

El resultado de la conversión tiene una resolución de 10 bits que se almacena en los registros ADCH y ADCL. La "justificación" del resultado puede ser hacia la izquierda o hacia la derecha, esto significa que el bit más significativo (msb), se encuentra en el msb del registro ADCH para el caso de justificación izquierda. En caso contrario, es decir en caso de justificación derecha, el bit menos significativo (lsb), se encuentra en lsb del registro ADCL. Para seleccionar uno de los dos casos de justificación, se debe modificar el bit ADLAR del registro ADMUX, es decir el bit 5 de dicho registro. Por defecto, el ATmega328P, está configurado para estar justificado a la derecha, con el valor 0 del bit ADLAR. El registro ADMUX configura los voltajes de referencia con los bits REFS1 Y REFS0 del registro ADMUX[7:6]; las configuraciones de estos bits se pueden ver en el Cuadro 8 ⁴.

³Es importante para activar la resistencia Pull-up, que el bit PUD del registro MCUCR se encuentre limpio, de lo contrario, se desactivaran las resistencias Pull-up internas.

⁴Tanto la selección 01 como 11, necesitan un capacitor en el pin AREF

REFS[1:0]	V_{ref}
00	Referencia en pin AREF
01	Referencia en pin AV_{CC}
10	Sin uso
11	Referencia de 1.1V

Cuadro 8: Selección de referencia en ATmega328P.

Para iniciar el módulo se puede utilizar el bit ADEN del registro ADCSRA, este bit activa el oscilador seleccionado con los bits ADPS[2:0] del registro ADCSRA[2:0], los factores de prescaler se presentan en el Cuadro 9.

ADPS[2:0]	Prescaler
000	2
001	2
010	4
011	8
100	16
101	32
110	64
111	128

Cuadro 9: Selección de frecuencia de oscilador de conversión.

2. Selección del oscilador de conversión

El oscilador se debe de seleccionar dependiendo del modo que se selecciona en los bits ADTS[2:0] del registro ADCSRB, únicamente cuando se activa el modo de activación automática *Auto Trigger* con el bit ADATE del registro ADCSRA. A continuación se presenta ambos modos de conversión.

Conversión única

En modo de conversión única, el prescaler que se selecciona solo afecta la velocidad de la conversión, la resolución del resultado es siempre de 10 bits. En este modo solo es necesario activar el bit ADSC del registro ADCSRA, el cual se limpia al finalizar la conversión, esto activa a su vez el bit ADIF⁵ del registro ADCSRA. Las conversiones en este modo se interrumpen hasta que la bandera se limpia y se vuelve a activar el bit ADSC.

⁵Esta bandera se activa independiente del estado de las interrupciones, con interrupciones activas o desactivadas.

Conversión de corrida libre

En modo de corrida libre, la frecuencia del oscilador afecta la resolución del resultado, de forma en la que independientemente del prescaler seleccionado, la conversión ocurre, para garantizar 10 bits de resolución en este modo, se debe de conseguir una frecuencia entre 50kHz y 200kHz, como se muestra en los cuadros 10 y 11, las combinaciones que logran esto con 8MHz, son los prescalers 128 y 64. Si se utilizan osciladores externos, se debe de verificar el cumplimiento mínimo de dicha frecuencia para garantizar 10 bits de resolución.

Prescaler	Frecuencia resultante	$T(\mu s)$	10 bits de resolución
2	8 MHz	0.125	NO
4	4 MHz	0.25	NO
8	2 MHz	0.5	NO
16	1 MHz	1	NO
32	500 kHz	2	NO
64	250 kHz	4	NO
128	125 kHz	8	SÍ

Cuadro 10: Cálculo de frecuencias del oscilador de conversión según prescaler para 16 MHz

Prescaler	Frecuencia resultante	$T(\mu s)$	10 bits de resolución
2	4 MHz	0.25	NO
4	2 MHz	0.5	NO
8	1 MHz	1	NO
16	500 kHz	2	NO
32	250 kHz	4	NO
64	125 kHz	8	SÍ
128	62500 Hz	16	SÍ

Cuadro 11: Cálculo de frecuencias del oscilador de conversión según prescaler para 8 MHz

3. Lectura en modo de conversión única sin interrupción

Al completar la configuración del módulo ADC mediante los registros ADMUX y ADCSRA, para este modo debe estar limpio el bit ADATE. Se puede iniciar la conversión en modo único mediante el bit ADSC del registro ADCSRA, al completar la conversión, el bit ADSC se limpia así como ADIF se activa, las conversiones se detienen hasta que ADIF se limpie. Posteriormente, se puede iniciar la conversión nuevamente.

```
CONF:
LDI    R16,    0b01100000 ;Canal 0 [2:0], Referencia en pin AVcc [7:6]
STS    ADMUX, R16         ;y Justificado hacia la izquierda[5].
LDI    R16,    0b10000111 ;Iniciamos el modulo ADC [7], Interrupcion
STS    ADCSRA, R16       ;desactivada [3], 128 de prescaler [2:0].
LDI    R16,    0b11000111 ;Se guarda la configuracion en variable R16
                                ;con el bit ADSC activo y ADIF limpio.
```

```

POLL:
STS    ADCSRA, R16          ;Se inicia la conversion
LDS    R17,    ADCSRA
SBRS   R17,    4            ;verifica la finalizacion de la conversion
RJMP   PC-3
RET

```

4. Lectura en modo de conversión única con interrupción

Posterior a realizar las configuraciones del módulo ADC mediante los registros ADMUX y ADCSRA, el bit ADATE debe estar limpio y ADIE debe estar activo. El vector de interrupción se encuentra en 0x002A. Se debe iniciar la conversión, mediante el bit ADSC, cada vez que se desee iniciar nuevamente.

```

.org 0x2A
RJMP   ADC

CONF:
SEI                                ;Activan interrupciones globales
LDI    R16,    0b01100000 ;Referencia en pin AVcc [7:6], canal 0 [2:0]
STS    ADMUX,  ADCSRA     ;justificado a la hacia la izquierda [5].
LDI    R16,    0b10001111 ;Iniciamos el modulo ADC [7], interrupcion
STS    ADCSRA, R16       ;activa [3], prescaler 128 [2:0].
LDI    R16,    0b11001111 ;Guarda la configuracion en R16 con el bit
                                ;ADSC activo y ADIF limpio.

ADC:
STS    ADCSRA, R16       ;Se limpia ADIF y se inicia conversion
RETI

```

5. Lectura en modo de corrida libre

Completando las configuraciones modificando los registros ADMUX y ADCSRA, en este último, activando el bit de ADATE. Se puede seleccionar el modo de Corrida libre desde el registro ADCSRB con los bits ADTS[2:0] con la combinación 000. Seleccionamos un prescaler de 128 para asegurarnos la resolución de 10 bits con 8 MHz del oscilador interno del ATmega328P. No hay necesidad de volver a activar el bit ADSC ni limpiar la bandera ADIF.

```

CONF:
LDI    R16,    0b01100000 ;Canal 0 [2:0], Referencia en pin AVcc [7:6]
STS    ADMUX,  R16        ;y justificado hacia la izquierda [5].
LDI    R16,    0b10100111 ;Iniciamos modulo [7], AutoTrigger activo [5]
STS    ADCSRA, R16       ;sin interrupcion [3], prescaler 128 [2:0]

START:

```

```

LDI    R16,    0b11100111 ;Activar las conversiones, solo es necesario
STS    ADCSRA, R16         ;activar el bit ADSC una vez.
RET

READ:
LDS    R17,    ADCL
LDS    R18,    ADCH         ;Guardamos resultados, siempre que se necesite.
RET

```

6. Lectura de múltiples entradas

Para realizar lecturas con múltiples entradas, se debe cambiar el canal en el registro ADMUX. Esto se realiza previo a iniciar una lectura, recomendando preestablecer la configuración de ADMUX en variables.

```

CONF:
LDI    R16,    0b11000111 ;Se guarda configuración para inicio
LDI    R17,    0b01100000 ;Configuración para canal 0 [2:0]
LDI    R18,    0b01100001 ;Configuración para canal 1 [2:0]
POLL:
STS    ADCSRA, R16         ;Inicia la conversion
LDS    R19,    ADCSRA
SBRS   ADCSRA, 4
RJMP   PC-3
RET
MAIN:
CALL   POLL
LDS    R20,    ADCH         ;Guardamos la lectura del canal 0 en R20
STS    ADMUX,  R18         ;Cambiamos a canal 1
CALL   POLL
LDS    R21,    ADCH         ;Guardamos la lectura del canal 1 en R21
STS    ADMUX,  R17         ;Cambiamos a canal 0
RJMP   MAIN

```

El código se ejecuta de forma más fluida si establecemos previamente las configuraciones en las variables, para solo tener que escribir sobre el registro ADMUX. Se puede realizar con la instrucción EOR, la cual realiza un *Exclusive Or* para solo modificar el canal y no alterar los demás registros, de esta forma se puede ahorrar una variable y en caso de hacerlo en interrupción, se puede aprovechar para establecer una rutina de cambio con una única instrucción.

```

.org 0x2A
RJMP   ADC

CONF:
LDI    R16,    0b11000111 ;Se guarda configuracion para inicio

```

```

LDI    R17,    0b00000001 ;Se guarda la mascara para el EOR

ADC:
LDS    ADCSRA, R18
SBRC   R18,    0
RJMP   PC+3
LDS    R20,    ADCH        ;Guardamos la conversion canal 0 en R20
RJMP   SWITCH
LDS    R21,    ADCH        ;Guardamos la conversion canal 1 en R21

SWITCH:
EOR    ADMUX,  R17        ;Instruccion de cambio de canal
STS    ADCSRA, R16        ;Se limpia ADIF y se inicia conversion
RETI

```

7. Resumen de registros utilizados

Registro	Usos
ADMUX	Este registro permite justificar el resultado y establecer las referencias a utilizar.
ADCSRA	En este registro se activa el módulo, se establece el prescaler a utilizar como frecuencia del reloj, además, permite activar las interrupciones e iniciar conversiones.

Cuadro 12: Resumen de registros utilizados en el módulo ADC en ATmega328P

C. Comparación módulos ADC: PIC16F887 y ATmega328P

Se compara la velocidad de conversión para ambos microcontroladores, para lo cual se utiliza el *logic analyzer*, se enciende un pin al momento de iniciar la conversión y se apaga al momento de finalizarla. Esto se realiza con el fin de obtener la velocidad de lectura y el tiempo entre lecturas.

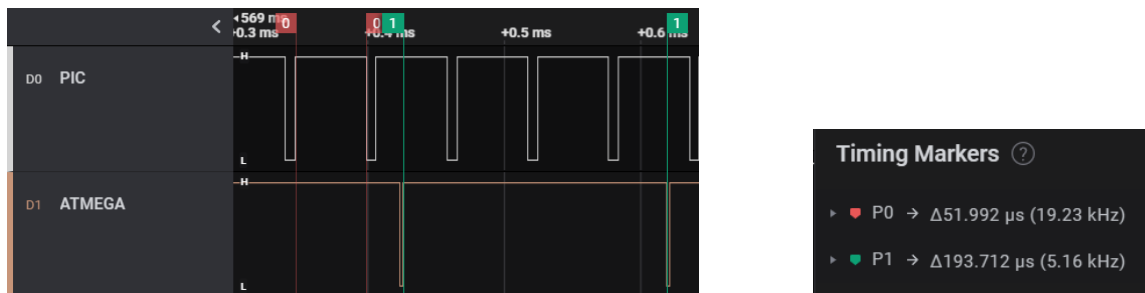


Figura 3: Comparación de tiempo de lecturas analógicas en ensamblador para PIC16F887 y ATmega328P.



Figura 4: Comparación de tiempo entre lecturas analógicas en ensamblador para PIC16F887 y ATmega328P.

Como se puede apreciar en Ensamblador, el ATmega328P tiene un mayor tiempo de lectura, contando con $193\mu s$ por lectura y un tiempo entre lecturas de $2\mu s$ con el oscilador interno a $8MHz$ y un prescaler de 32, logrando un oscilador de conversión equivalente a $250kHz$. Se configura con los mismos valores el PIC16F887, obteniendo un tiempo de lectura de $51\mu s$ y un tiempo entre lecturas de $7\mu s$.

El resultado del PIC es comparable con lo esperado según el Cuadro 6, debido a los aproximadamente 13 ciclos de reloj de conversión como se aprecia en la ecuación 1, sin embargo, en el caso del ATmega328P, no se obtiene el resultado esperado, presumiblemente por un código aún ineficiente a comparación del obtenido en el PIC16F887.

$$4\mu s(13T_{AD}) = 52\mu s \quad (1)$$

Los tiempos entre lectura aumentan considerablemente entre el PIC y el ATmega328P. El PIC16F887 demora más tiempo entre lecturas debido a la necesidad del microcontrolador de restablecer la información previa a la interrupción generada por el ADC, esto le representa una diferencia de $5\mu s$ de desventaja contra el ATmega328P.

Los resultados obtenidos en C se pueden observar en las figuras 5 y 6, ambos configurados a $8MHz$ de oscilador interno con prescaler de 32, obteniendo de esta forma un oscilador de conversión equivalente a $250kHz$, es decir, $4\mu s$ por ciclo de reloj.

En el caso del PIC16F887 se requiere un tiempo aproximado de 13 Tiempos de adquisición. Según la información proporcionada por el Cuadro 6, el tiempo de adquisición es de $4\mu s$, por lo que el tiempo de la conversión es de $52\mu s$ teóricamente, contrastado contra el tiempo obtenido por las mediciones las cuales nos dan un resultado de $67\mu s$, obteniendo un resultado cercano en C, el error se debe que C no realiza únicamente las instrucciones mínimas necesarias. Naturalmente, los códigos en C, son menos eficientes que los códigos en Ensamblador.

En el caso del ATmega328P, se requiere un tiempo de 13 ciclos de reloj (oscilador de conversión), por lo que el valor esperado en este caso es de $52\mu s$, el tiempo de conversión obtenido por los resultados es de aproximadamente $56\mu s$, obteniendo un error de $4\mu s$.

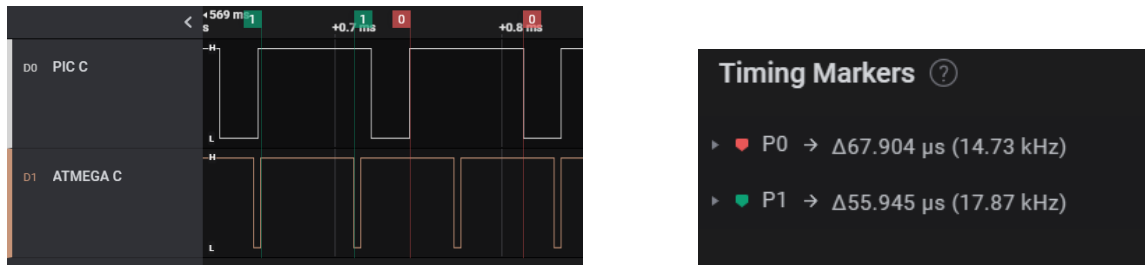


Figura 5: Comparación de tiempo de lecturas analógicas en C para PIC16F887 y ATmega328P.

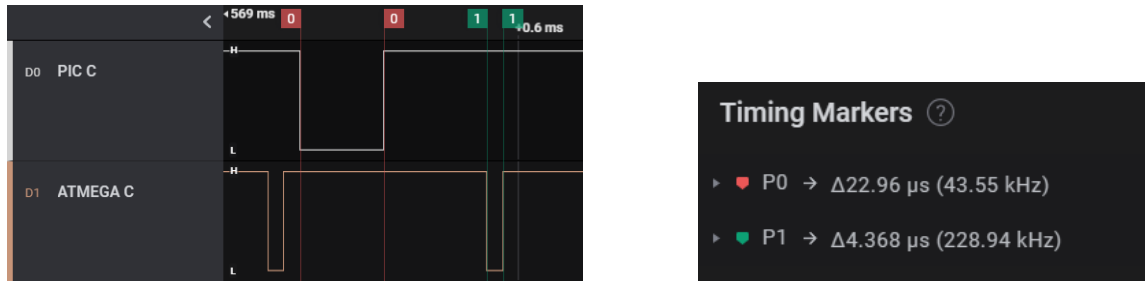


Figura 6: Comparación de tiempo entre lecturas analógicas en C para PIC16F887 y ATmega328P.

La diferencia de tiempo entre lectura se vuelve más notoria en C, como se puede apreciar en la figura 6. En la cual el PIC demora $23\mu s$ entre las lecturas y el ATmega328P solo demora $4\mu s$. Cabe mencionar que la precisión de los datos obtenidos en el ATmega328P era mejor que la del PIC, esto evidenciado con el programa realizado en la que visualmente de una resolución de 10 bits se presentaban los 8 más significativos, variando continuamente los últimos 4 bits de resolución del PIC, a diferencia del ATmega328P para el cual no variaba los 8 bits más significativos.

1. Comparación de cantidad de instrucciones y tiempo de ejecución de códigos

Un ciclo de instrucción en el PIC16F887 requiere 4 periodos de reloj, por lo que en el Cuadro 13, se puede observar la última columna con los periodos de reloj utilizados en la configuración del módulo ADC. Con un total de 192.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción	Periodos de reloj
GOTO	1	2	2	8
MOVWF	7	1	7	28
CALL	4	2	8	32
CLRF	8	1	8	32
MOVLW	7	1	7	28
RETURN	4	2	8	32
BANKSEL	8	1	8	32
Total	39			192

Cuadro 13: Conteo de instrucciones en la configuración del ADC y periodos de reloj totales utilizados en PIC16F887, según programa propuesto, en ensamblador.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción	Periodos de reloj
GOTO	5	2	10	40
MOVWF	7	1	7	28
SWAPF	4	1	4	16
BTFSS	2	2	4	16
MOVF	4	1	4	16
BCF	3	1	3	12
BSF	4	1	4	16
NOP	7	1	7	28
RETFIE	1	2	2	8
Total	37			180

Cuadro 14: Conteo de instrucciones en la ejecución del ADC y periodos de reloj totales utilizados en PIC16F887, según programa propuesto, en ensamblador.

Un ciclo de instrucción en el ATmega328P, así como los dispositivos de AVR, pueden demorar de 1 a 4 ciclos de máquina, esto significa que no hay necesidad de convertir entre ciclos de instrucción y ciclos de máquina, tal como se presenta en los cuadros 15 y 16.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción
CALL	3	4	12
LDI	15	1	15
OUT	4	1	4
STS	2	2	4
RET	3	4	12
TOTAL	27		47

Cuadro 15: Conteo de instrucciones en la configuración del ADC y ciclos de instrucción totales utilizados en ATmega328P, según programa propuesto, en ensamblador.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción
CALL	4	4	16
LDS	4	1	4
OUT	2	1	2
STS	4	2	8
RET	4	4	16
RJMP	3	2	6
SBRS	2	3	6
NOP	12	1	12
TOTAL	35		70

Cuadro 16: Conteo de instrucciones en la ejecución del ADC y ciclos de instrucción totales utilizados en ATmega328P, según programa propuesto, en ensamblador.

Se puede observar que tanto realizar la configuración, como ejecutar la tarea de lectura, con canales alternados conlleva más ciclos de máquina en el PIC16F887 que en el

ATmega328P. Tal como se puede apreciar en el Cuadro 17, el total de ciclos de reloj en la configuración del PIC es de 192 contra la configuración del ATmega328P con 47. De la misma forma, ejecutar la lectura con cambio de canal requiere de 180 ciclos de reloj en el PIC contra los 70 del ATmega328P.

Microcontrolador	Configuración	Ejecución
PIC16F887	192	180
ATmega328P	47	70

Cuadro 17: Comparación ciclos de reloj para ADC en ensamblador.

El ATmega328P, tiene una gran ventaja sobre el PIC16F887, puesto que solo necesita una instrucción para colocar valores, si se cargan en variables las configuraciones, se puede cargar directamente utilizando las instrucciones STS, en cambio en PIC se debe de cargar el valor a W para posteriormente utilizar MOVWF, lo que requiere 2 instrucciones, por lo tanto los códigos son más extensos. En términos de ciclos de reloj, el ATmega328P presenta una ventaja contra el PIC, puesto que en el PIC todas las instrucciones utilizan 4 ciclos de reloj. En cambio, como se explica anteriormente el ATmega328P puede utilizar entre 1 a 4 ciclos de reloj por instrucción. El ATmega328P es mucho más rápido en ejecuciones gracias a esta característica.

Con respecto a la cantidad de instrucciones necesarias para completar la configuración y ejecución del módulo, se presenta una comparación en el Cuadro 18. El ATmega328P necesita 12 instrucciones menos para la configuración y requiere de 2 menos por ejecución.

Microcontrolador	Configuración	Ejecución
PIC16F887	39	37
ATmega328P	27	35

Cuadro 18: Comparación cantidad de instrucciones necesarias para configuración y ejecución del módulo ADC en ensamblador.

2. Comparación de cantidad de registros utilizados por módulo

Los registros utilizados por cada módulo se presentan a continuación, se obvia en el caso del PIC16F887 los registros de interrupciones INTCON, PIE1 Y PIR1, además, los registros de configuración de los pines TRISA, TRISB y TRISE. Obteniendo un total de registros necesarios en el PIC16F887 equivalente a 4 y 2 registros en el ATmega328P.

Microcontrolador	Registros
PIC16F887	ANSEL ANSELH ADCON0 ADCON1
ATmega328P	ADMUX ADCSRA

Cuadro 19: Comparación de cantidad de registros utilizados por módulo

Módulo de Modulación de Ancho de Pulso (PWM)

Tanto el PIC16F887 como el ATmega328P, tienen un módulo de modulación de ancho de pulso también conocido como PWM por sus siglas en inglés. Se comparan ambos módulos en función de la cantidad de registros que se modifican y los ciclos de reloj que conlleva su configuración y ejecución. Se realiza el programa en Ensamblador y C para ambos microcontroladores. Se observa el funcionamiento y respuesta del microcontrolador utilizando el *logic analyzer*, se configura la lectura de datos a 125MS/s por 5s. A continuación se presenta la comparación de ambos módulos. A continuación se explica el módulo PWM de ambos módulos.

A. Módulo PWM en PIC16F887

1. Registros y funcionamiento

El microcontrolador cuenta con 2 canales PWM, cada uno de estos utiliza su respectivo registro de control CCP1CON y CCP2CON, en los cuales se puede seleccionar el modo de operación del módulo con los bits CCPxM [3:0]¹, tal como se puede observar en el Cuadro 21. En el caso del canal 1, es un canal mejorado, por lo que se puede manejar 4 salidas ubicadas según se presenta en el Cuadro 20; a diferencia del canal 2, el cual solo maneja la salida CCP2 ubicada en el puerto RC1.

Salida	P1A	P1B	P1C	P1D
Pin	RC2	RD5	RD6	RD7

Cuadro 20: Salidas del canal 1 de PWM mejorado

¹Donde x se refiere a 1 o 2, dependiendo del canal PWM utilizado.

Modo	CCP1M
P1A, P1B, P1C y P1D activos en HIGH	1100
P1A, P1C activos en HIGH; P1B, P1D activos en LOW	1101
P1A, P1C activos en LOW; P1B, P1D activos en HIGH	1110
P1A, P1B, P1C y P1D activos en LOW	1111

Cuadro 21: Modos del canal 1 de PWM mejorado

La configuración del PWM requiere del timer 2, por lo que se debe configurar el registro T2CON y PR2. T2CON permite configurar el postscaler y prescaler del timer, sin embargo, en modo de PWM no se utiliza el postscaler, así como no se debe activar el bit de encendido del timer 2; PR2 define el periodo del timer, por lo tanto la frecuencia del PWM. El ciclo de trabajo se modifica por medio del registro CCPRxL y los bits 4:5 del registro CCPxCON.

Se tienen ecuaciones para determinar los valores necesarios, se presentan en las ecuaciones 2 y 3.

$$PR2 = \frac{T_{deseada}}{(T_{osc} * Prescaler * 4)} - 1 \quad (2)$$

$$CCPRxL = D.C. * (PR2 + 1)/25 \quad (3)$$

Donde PR2 es el registro de comparación del timer2 y CCPRxL es el registro que establece el ciclo de trabajo del PWM, ²D.C. simboliza el ciclo de trabajo, T_{osc} siendo el periodo de oscilación, es decir el inverso multiplicativo de la Frecuencia de oscilación y $T_{deseada}$ siendo el periodo deseado en segundos.

```
PWM1_set:
BANKSEL TRISC
CLRF    TRISC
CLRF    TRISA
MOVLW  0b01111111
MOVWF  PR2
BANKSEL PORTC
CLRF    PORTC
MOVLW  0b00111100
MOVWF  CCP1CON
MOVLW  0b01111100
MOVWF  CCPR1L
MOVLW  0b00000111
MOVWF  T2CON
RETURN
```

²En la carpeta GITHUB del proyecto se adjunta un excel calculadora de PWM para facilitar los cálculos.

Se establece el periodo del timer 2 con PR2, se configura el puerto C como salidas y se limpia. Se establecen las configuraciones del PWM1 desde el registro CCP1CON y el ciclo de trabajo desde el registro CCPR1L. La configuración que se obtiene a partir de esta programación es una salida única en el pin RC2 [7:6], y se configura que el activo sea en HIGH para los 4 canales del PWM1 (esta configuración se puede modificar a 1101 en los bits [3:0], debido que solo estamos utilizando el canal P1A).

2. Resumen de registros utilizados

Registro	Usos
PR2	Determina el periodo del PWM.
CCP1CON	Permite seleccionar las salidas activas y el modo en el que se utilizará el módulo.
CCPR1L	Este registro determina el ciclo de trabajo.
T2CON	En este registro se activa el timer 2 y se configura el prescaler que se utilizará.

Cuadro 22: Registros utilizados en el módulo PWM en PIC16F887

B. Módulo PWM en ATmega328P

1. Registros y funcionamiento

El microcontrolador cuenta con 6 canales PWM, configurables con timers de 8 y 16 bits, modificando los registros TCCRxA y TCCRxB, siendo estos los registros de configuración de funcionamiento.

Los bits [7:6] del registro TCCRxA, varían dependiendo de la función que cumple el módulo, en este caso analizaremos en PWM normal, por lo que su configuración se puede apreciar en el Cuadro 24, estos bits modifican únicamente el funcionamiento del canal A, por lo que solo se modifica el pin D6, los pines que afectan cada canal se pueden encontrar en el Cuadro 23, los bits [5:4] del registro TCCRxA modifican el canal B del PWM, las configuraciones del mismo se pueden apreciar en el Cuadro 25. Por último, el bit [3] del registro TCCRxB y los bits [1:0] del registro TCCRxA seleccionan la función que cumple el módulo. Las opciones disponibles se presentan en el Cuadro 26.

Canal	OC0A	OC0B	OC1A	OC1B	OC2A	OC2B
Pin	PD6	PD5	PB1	PB2	PB3	PD3

Cuadro 23: Canales y pines correspondientes

COM0A1	COM0A0	Función	
0	0	Operación normal del pin D6, módulo desconectado	
0	1	WGM02 = 0: Operación normal del pin D6	WGM02 = 1: Invertir estado actual de la salida módulo OC0A (PD6)
1	0	Limpiar OC0A cuando se alcanza el valor del comparador.	
1	1	Establecer HIGH en OC0A cuando se alcanza el valor del comparador.	

Cuadro 24: Configuración canal A de PWM 0

COM0B1	COM0B0	Función	
0	0	Operación normal del pin D5, módulo desconectado	
0	1	Reservado	
1	0	Limpiar OC0B cuando se alcanza el valor del comparador.	
1	1	Establecer HIGH en OC0B cuando se alcanza el valor del comparador.	

Cuadro 25: Configuración canal B de PWM 0

WGM 2	WGM [1:0]	Función del módulo
0	00	Normal
0	01	PWM, corrección de fase
0	10	Limpiar timer al comparar (CTC)
0	11	Fast PWM
1	00	Reservado
1	01	PWM, corrección de fase modo OCR0A
1	10	Reservado
1	11	Fast PWM modo OCR0A

Cuadro 26: Funciones del módulo

Los bits [7:6] del registro TCCRxB son bits de solo escritura, siempre se leen como 0. Estos bits no se utilizan en el modo PWM. El bit 3 se mencionó anteriormente, este es el bit WGM2 visto en el Cuadro 26. Por último los bits [2:0] del registro TCCRxB establecen el reloj que se estará utilizando para el módulo, tal como se puede apreciar en el Cuadro 27.

CS[2:0]	Función del módulo
000	Sin reloj
001	CLK sin prescaler
010	CLK/8 con prescaler
011	CLK/64 con prescaler
100	CLK/256 con prescaler
101	CLK/1024 con prescaler
110	Reloj externo en PD4, flanco negativo
111	Reloj externo en PD4, flanco positivo

Cuadro 27: Selección de reloj para módulo PWM

2. Cálculos de PWM

Obtener los periodos de PWM depende del modo que se selecciona de entre los disponibles, según los modos disponibles en el Cuadro 26, existen 4 modos cada uno con su modo invertido (Cuadro 24). De otra forma se puede colocar un oscilador en el pin D4 para obtener la frecuencia del PWM exacta necesaria.

PWM Corrección de fase

En corrección de fase normal se tiene definido el periodo del PWM en función de la Frecuencia del microcontrolador y los 5 prescaler definidos en el Cuadro 27. La ecuación para determinar cuál es la frecuencia del PWM actual está dada por la ecuación 4.

$$F_{PWM} = \frac{F_{osc}}{N * 510} \quad (4)$$

Donde N es el prescaler seleccionado y F_{osc} es la frecuencia de oscilación. Como se puede observar en la ecuación, no hay muchos valores disponibles en frecuencias de PWM debido a los valores de N limitados. Para modificar el ciclo de trabajo de este modo, solo es necesario multiplicar 255 por el porcentaje de ciclo de trabajo a utilizar, tal como se aprecia en la ecuación 5, el resultado se escribe en el registro OCR0A, el cual modificará el ciclo de trabajo en este modo.

$$OCR0A = 255(\%D.C.) \quad (5)$$

Donde D.C. significa el ciclo de trabajo representado en porcentaje.

Fast PWM normal

En modo Fast PWM, se define el periodo del PWM de la misma forma que en el modo de corrección de fase normal, estando limitados a la frecuencia del oscilador y los 5 valores de prescaler disponibles. La ecuación 6 presenta el cálculo de periodo y la ecuación 5 presenta el número a cargar al registro OCR0A dependiendo del ciclo de trabajo solicitado, siendo el mismo cálculo que el empleado en el modo de corrección de fase.

$$F_{PWM} = \frac{f_{osc}}{N * 256} \quad (6)$$

PWM corrección de fase modo OCR0A

A diferencia del modo PWM corrección de fase normal, este modo permite obtener diferentes frecuencias a partir de modificar el valor tope del PWM ahora ubicado en OCR0A y modificar el ciclo de trabajo desde el registro OCR0B, permitiendo obtener diferentes frecuencias en el PWM. La ecuación 7 permite obtener el periodo del PWM.

$$F_{Timer} = \frac{F_{osc}}{N * 510}$$

$$OCR0A = 255 * \frac{F_{Timer}}{F_{deseada}} \quad (7)$$

Donde N es el prescaler. El valor obtenido en OCR0A será el fin del periodo del PWM, por lo que el valor en el registro para ciclo de trabajo, deberá ser menor que este. La ecuación 8 presenta el valor que se debe cargar en el registro OCR0B, este valor depende del obtenido previamente para determinar el máximo del timer.

$$OCR0B = OCR0A * \%D.C. \quad (8)$$

Fast PWM modo OCR0A

A diferencia del modo Fast PWM normal, este modo permite obtener diferentes frecuencias a partir de modificar el valor tope del PWM ahora ubicado en OCR0A y modificar el ciclo de trabajo desde el registro OCR0B, permitiendo obtener diferentes frecuencias en el PWM, funcionando de la misma forma que el modo OCR0A del PWM corrección de fase. La ecuación 9 permite obtener el periodo del PWM.

$$F_{Timer} = \frac{F_{osc}}{N * 256}$$

$$OCR0A = 255 * \frac{F_{Timer}}{F_{deseada}} \quad (9)$$

Donde N es el prescaler. El valor obtenido en OCR0A será el fin del periodo del PWM, por lo que el valor en el registro para ciclo de trabajo, deberá ser menor que este. La ecuación 8 presenta el valor que se debe cargar en el registro OCR0B, este valor depende del obtenido previamente para determinar el máximo del timer, siendo la misma ecuación utilizada en el caso anterior.

3. Código de ejemplo, Fast PWM normal

```
PWM_SET:
LDI    R16,    0b00100000 //PIN D5 es salida
OUT    DDRD,   R16
LDI    R16,    0b10000011 //Limpiar al alcanzar
OUT    TCCR0A, R16        //OCR0A [7:6], Fast PWM [1:0]
LDI    R16,    0b00000011 //WGM2 = 0 [3], CLK /64 [2:0]
OUT    TCCR0B, R16
LDI    R16,    0x80        //duty 50%
OUT    OCR0A,  R16
```

Se establece el pin D5 como salida, se selecciona el modo Fast PWM normal y se establece un prescaler de 64, además de un ciclo de trabajo de 50% inicialmente.

4. Resumen de registros utilizados

Registro	Usos
TCCR0A	Este registro permite configurar las salidas A y B del módulo y seleccionar la forma en la que funciona el PWM
TCCR0B	Este registro permite configurar la frecuencia de oscilación del módulo.
OCR0A	Es el registro de comparación, al momento en el que se iguala, en modo corrección de fase marca el periodo, en los demás modos, marca el ciclo de trabajo.
OCR0B	En modo corrección de fase este registro marca el Ciclo de trabajo, en otros funciona como el periodo.

Cuadro 28: Registros utilizados en el módulo PWM en ATmega328p

C. Comparación módulos PWM: PIC16F887 y ATmega328P

Se compara el funcionamiento de ambos módulos por medio de la cantidad de ciclos de máquina utilizados en cada uno de los programas, además, se observa el comportamiento de los módulos haciendo uso del analizador lógico. Esto para verificar que cumplan los tiempos requeridos, como se puede apreciar en la Figura 7. Ambos microcontroladores con una frecuencia de 488.28125Hz.³ Los datos extraídos a partir de las capturas se pueden apreciar en el Cuadro 29. Donde el PIC consigue un error del 0.13% y el ATmega328P un error del 2.4%.



Figura 7: Comparación de respuesta del módulo PWM en ensamblador para PIC16F887 y ATmega328P

³La frecuencia es seleccionada a partir de una de las frecuencias obtenidas de forma predeterminada por el ATmega328P en sus modos normales, en este caso, del Fast PWM normal, el PIC puede obtenerla con configuraciones estándar, con 0xFF en PR2 y un prescaler de 16.

Microcontrolador	Frecuencia obtenida	Error
PIC 16F887	487.628 Hz	0.13 %
ATmega328P	476 Hz	2.45 %

Cuadro 29: Medición de error de frecuencias obtenidas de los PWM de ambos microcontroladores

1. Comparación de tamaño y tiempo de ejecución de códigos

Se comparan las instrucciones necesarias para la configuración de los códigos, tal como se puede observar en los Cuadros 30 y 32. Además de comparar las instrucciones necesarias para cambiar el ciclo de trabajo, tal como se presentan en los Cuadros 31 y 33.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción	Periodos de reloj
GOTO	1	2	2	8
MOVWF	6	1	6	24
MOVLW	6	1	6	24
CALL	2	2	4	16
CLRF	2	1	2	8
BANKSEL	2	1	2	8
RETURN	2	2	4	16
Total	21			104

Cuadro 30: Conteo de instrucciones en la configuración del PWM y periodos de reloj totales utilizados en PIC16F887, según programa propuesto en ensamblador.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción	Periodos de reloj
MOVWF	1	1	1	4
MOVLW	1	1	1	4
Total	2			8

Cuadro 31: Conteo de instrucciones en el cambio del ciclo de trabajo del PWM y periodos de reloj totales utilizados en PIC16F887, según programa propuesto en ensamblador.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción
CALL	2	4	8
LDI	4	1	4
OUT	4	1	4
RET	2	4	8
Total	12		24

Cuadro 32: Conteo de instrucciones en la configuración del PWM y ciclos de instrucción totales utilizados en ATmega328P, según programa propuesto en ensamblador.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción
LDI	1	1	1
OUT	1	1	1
Total	2		2

Cuadro 33: Conteo de instrucciones en el cambio del ciclo de trabajo del PWM y ciclos de instrucción totales utilizados en ATmega328P, según programa propuesto en ensamblador.

Para el conteo de instrucciones se toma únicamente en cuenta las configuraciones relacionadas con el PWM y para el programa de ejecución solo se toma en cuenta el cambio de ciclo de trabajo, sin considerar el método del cuál recibe el número. Cabe resaltar del ATmega328P que se puede beneficiar de tiempos aún menores si se han cargado previamente los ciclos de trabajo a utilizar en los registros R, ahorrando tiempo de ejecución por vuelta en la instrucción LDI de la modificación de ciclo de trabajo. La comparación de ciclos de reloj, se presenta en el Cuadro 34 y como se puede apreciar, el ATmega328P se ejecuta la configuración de forma mucho más rápida que el PIC16F887, con una diferencia de 80 ciclos de reloj. Esto se puede notar incluso más evidentemente al analizar la cantidad de instrucciones para configuración por cada uno (Cuadro 35).

Microcontrolador	Configuración	Ejecución
PIC16F887	104	8
ATmega328P	24	2

Cuadro 34: Comparación ciclos de reloj para PWM en ensamblador.

Microcontrolador	Configuración	Ejecución
PIC16F887	21	2
ATmega328P	12	2

Cuadro 35: Comparación de cantidad de instrucciones necesarias para configuración y ejecución del módulo PWM en ensamblador.

La principal diferencia es la cantidad de instrucciones que requiere mover literales a registros en el PIC16F887, así como tener que cambiar de banco seleccionado. El ATmega328P no necesita ninguna de estas operaciones, otorgándole una gran ventaja al momento de minimizar las instrucciones utilizadas, así como ciclos de máquina invertidos.

2. Comparación de cantidad de registros utilizados por módulo

Los registros utilizados por cada módulo se presentan a continuación. En este módulo ambos utilizan 4 registros para configurarse y operar.

Microcontrolador	Registros
PIC16F887	PR2 CCP1CON CCPR1L T2CON
ATmega328P	TCCR0A TCCR0B OCR0A OCR0B

Cuadro 36: Comparación de cantidad de registros utilizados por módulo

Módulo de Recepción y Transmisión Asíncrona Universal (UART)

Tanto el PIC16F887, como el ATmega328P; cuentan con al menos un módulo UART que permite la comunicación serial bidireccional haciendo uso de 2 vías, la transmisión y recepción, usualmente nombradas como RX y TX. Coordinadas por un baudaje (baud rate) que indica la cantidad de símbolos enviados por segundo. Se comparan ambos módulos por medio de la cantidad de registros utilizados y de instrucciones utilizados en configuración como operación, en cada uno de los módulos. Al ser un protocolo de comunicación, no se utiliza el analizador lógico, debido a que esencialmente, las señales son idénticas.

A. Módulo UART en PIC16F887

1. Registros y funcionamiento

El módulo puede funcionar de forma síncrona o asíncrona, esto se configura por medio del bit SYNC, el cual es el bit 4 del registro TXSTA, se puede configurar también la cantidad de bits que se envían por cada transmisión por medio del bit TX9, se pueden seleccionar 8 o 9 bits, si se activa el bit 6 habilita la transmisión de 9 bits. La comunicación permite ser configurada para solo recibir o solo transmitir, por lo que de no ser necesaria alguna de las funciones se puede desactivar en sus respectivos registros. El bit TXEN (bit 5) del registro TXSTA permite habilitar la transmisión. En caso de ser Asíncrona la comunicación, se puede establecer un baudaje de alta velocidad o baja velocidad, por medio del bit 2 del registro TXSTA.

En caso de necesitarse en la comunicación la recepción de datos, se pueden modificar los bits SREN o CREN, los cuales funcionan para recepción única o recepción continua respectivamente. Ambos se encuentran en el registro RCSTA, siendo los bits 5 y 4. El bit SREN es irrelevante en modo asíncrono, en este modo solo es necesario establecer si se activa o desactiva la recepción por medio del bit CREN (bit 4). En el registro RCSTA se activa el puerto serial por medio del bit 7, SPEN. Además, permite configurar la recepción de 8 o 9 bits por medio del bit 6, RX9.

La configuración del Baudrate se lleva a cabo desde el registro BAUDCTL. Este registro permite seleccionar si el generador de baudrate es de 8 o 16 bits, esta selección se realiza por medio del bit 3, BRG16. Se debe activar el bit para seleccionar el generador de baudrate de 16 bits.

2. Configuración y cálculo de Baudrate

Dependiendo de la configuración de los bits SYNC, BRG16 y BRGH, la ecuación para obtener el baudrate varía. Tal como se presenta en el Cuadro 37.

Configuración			Baudrate Generator / modo	Ecuación para Baudrate
SYNC	BRG16	BRGH		
0	0	0	8 bits / asíncrono	$\frac{F_{osc}}{[64(n+1)]}$
0	0	1	8 bits / asíncrono	$\frac{F_{osc}}{[16(n+1)]}$
0	1	0	16 bits / asíncrono	
0	1	1	16 bits / asíncrono	$\frac{F_{osc}}{[4(n+1)]}$
1	0	x	8 bits / síncrono	
1	1	x	16 bits / síncrono	

Cuadro 37: Ecuaciones para configuración del baudrate en PIC16F887

Donde n es el registro de 8 o 16 bits, dependiendo del seleccionado, que determina el valor de SPBRG, es decir los registros SPBRGH:SPBRG. Por practicidad es mejor fijar un Baudrate y despejar de la ecuación para obtener n. Tal como se presentan en las ecuaciones siguientes.

Cálculo de SPBRG para UART de baja velocidad asíncrono de 8 bits

$$SPBRG = \frac{F_{osc}}{BaudRate} - 1 \quad (10)$$

Cálculo de SPBRG para UART de alta velocidad asíncrono de 8 bits y UART de baja velocidad asíncrono de 16 bits

$$SPBRG = \frac{F_{osc}}{BaudRate} - 1 \quad (11)$$

Cálculo de SPBRG para UART de alta velocidad asíncrono de 16 bits y UART síncrono

$$SPBRG = \frac{F_{osc}}{4 \cdot BaudRate} - 1 \quad (12)$$

```

UART_Tset:
    BANKSEL SPBRG
    MOVLW    0x00
    MOVWF    SPBRGH
    MOVLW    0b00001100
    MOVWF    SPBRG    ;12 en SPBRG para 9600
    MOVLW    0b00000010    ;8 bits/transmision[6], Asíncrono [4]
    MOVWF    TXSTA    ;Baudrate lento [2]
    BANKSEL BAUDCTL
    MOVLW    0b01000000    ;8 bits Baudrate [3]
    MOVWF    BAUDCTL
    BANKSEL RCSTA
    MOVLW    0b10000000    ;Puerto serial activado [7]
    MOVWF    RCSTA
    BANKSEL TXSTA
    BSF     TXSTA, 5    ;Se activan las transmisiones.
    
```

3. Envío de datos

Se debe esperar que la bandera TRMT del registro TXSTA se apague, al momento en el que la bandera está en 0, se puede escribir el dato a transmitir al registro TXREG.

```

Transmitir:
    BTFSC    TXSTA, 1    ;Se revisa el estado del bit TRMT
    GOTO     $-1
    MOVF     Dato, W    ;Se mueve el dato a enviar a W
    MOVWF    TXREG    ;Se inicia la transmisión.
    
```

4. Recepción de datos

Al momento en el que se completa la recepción de un carácter, se activa la bandera RCIF, por lo que se puede verificar continuamente el estado, o habilitar interrupciones para que guarden el valor de RCREG. Ambas opciones son posibles al recibir un dato.

Recibir:

```
BTFSS   PIR1,    5    ;Se verifica el estado de la bandera
GOTO    $-1
MOVF    RCREG,  W    ;Se mueve la recepción en W
MOVWF   DatoRec     ;Se guarda la recepción en una variable
```

5. Resumen de registros utilizados

Registro	Usos
BAUDCTL	Este registro permite seleccionar determinar si el generador de baudrate es de 16 o 8 bits.
TXSTA	Este registro permite habilitar la transmisión, seleccionar si se desea modo síncrono o asíncrono y seleccionar la velocidad del Baudrate.
RCSTA	Este registro cuenta con la habilitación del módulo y habilitar la recepción de un dato o continua.
SPBRGH	Estos registros guardan el periodo calculado para obtener un determinado Baudrate.
SPBRG	
RCREG	Es el buffer que guarda los datos recibidos en UART.
TXREG	Este registro es el que sostiene el dato a enviar en UART.

Cuadro 38: Registros utilizados en el módulo UART en PIC16F887

B. Módulo UART en ATmega328P

1. Registros y funcionamiento

El módulo puede funcionar de forma síncrona o asíncrona, esto se configura por medio de los bits UMSEL0 [1:0] del registro UCSR0C, tal como se puede apreciar en el Cuadro 39, además, se puede configurar también la cantidad de bits que se envían por cada transmisión por medio de los bits UCSZ0 [2:0] de los cuales, los bits [1:0] forman parte del registro UCSR0C [2:1] y el bit 2 del UCSZ0, forma parte del registro UCSR0B [2]. Se pueden seleccionar de 5 a 9 bits, la configuración de bits se presenta en el Cuadro 40.

UMSEL0[1:0]	Modo
00	Asíncrono
01	Síncrono
10	reservado
11	Master SPI

Cuadro 39: Modos disponibles del módulo UART

UCSZ02	UCSZ0[1:0]	Tamaño
0	00	5 bits
0	01	6 bits
0	10	7 bits
0	11	8 bits
1	11	9 bits

Cuadro 40: Configuraciones para modificar tamaño de carácter

La comunicación permite ser configurada para solo recibir o solo transmitir, esto se logra por medio de los bits RXEN0 (Recieve enable) y TXEN0 (Transmit enable), estos corresponden a los bits [4] y [3] del registro UCSR0B. Por lo que de no ser necesaria alguna de las funciones se puede desactivar en sus respectivos registros. El módulo permite establecer alta velocidad o baja velocidad, por medio del bit U2X0, el bit 1 del registro UCSR0A. En el registro UCSR0C se puede modificar a su vez, la cantidad de bits de parada, siendo limpio para seleccionar 1 bit de parada y activo para seleccionar 2 bits de parada; Se puede seleccionar si se requiere bit de paridad por medio de los bits UPM0[1:0], tal como se aprecia en el Cuadro 41.

UPM0[1:0]	Tamaño
00	Desactivado
01	Reservado
10	Habilitado, par
11	Habilitado, impar

Cuadro 41: configuración de Bit de paridad

En el registro UCSR0B se puede seleccionar las interrupciones que se desean activar, con los bits RXCIE0, TXCIE0 Y UDRIE0, funcionando como activador de interrupciones de recepción, transmisión y registro de datos vacío.

El registro UCSR0A funciona como el registro de banderas del módulo, indicando si se termina la recepción (Bit 7, RXC0), si se completa la transmisión (Bit 6, TXC0) y si el buffer de datos está vacío.

El registro para modificación del baudrate, es el registro UBRR0, la forma en la que se calcula este valor se presenta en el Cuadro 42. El mismo depende de la configuración de otros bits, tal como se presenta en el Cuadro.

Configuración		Ecuación
UMSEL0[1:0]	U2X0	
00	0	$\frac{F_{osc}}{16 * Baud} - 1$
00	1	$\frac{F_{osc}}{8 * Baud} - 1$
01	x	$\frac{F_{osc}}{2 * Baud} - 1$

Cuadro 42: Ecuaciones para cálculo del registro UBRR0

Código propuesto de configuración

```

UART_set:
LDI R16,0b00000000 ;Baudrate de 9615
LDI R17,0b00110011 ;Se coloca 51 en los registros UBRR0H:UBRR0L
STS UBRR0H,R16
STS UBRR0L,R17
LDI R16,0b00001000 ;Recepcion apagada [4], transmision activa [3], interrupciones ap
LDI R17,0b00000110 ;Asincrono [7:6], sin paridad [5:4], 1 stop [3], 8 bits [2:1], x
STS UCSR0B,R16
STS UCSR0C,R17

```

2. Envío de datos

Se debe esperar que la bandera TXC0 del registro UCSR0A se apague, al momento en el que la bandera está en 0, se puede escribir el dato a transmitir al registro UDR0.

```

UART_tx:
LDS R16,UCSR0A
SBRS R16,5
RJMP UART_tx
STS UDR0,R18
RET

```

3. Recepción de datos

Al momento en el que se completa la recepción de un carácter, se activa la bandera RXC0, por lo que se puede verificar continuamente el estado, o habilitar interrupciones para que guarden el valor de UDR0. Ambas opciones son posibles al recibir un dato.

```

UART_rx:
    LDS R16,UCSROA
    SBRS R16,7
    RJMP UART_rx
    LDS R18,UDRO
    RETI

```

4. Resumen de registros utilizados

Registro	Usos
UCSR0A	Este registro funciona como indicador del estado del módulo. Indica cuando una transmisión o recepción se ha completado, cuando el buffer de datos está vacío y como configuración de velocidad de Baudrate.
UCSR0B	En este registro se puede habilitar interrupciones, además, permite habilitar la recepción o transmisión de datos y establecer la cantidad de bits por comunicación.
UCSR0C	Este registro permite configurar si se desea que la comunicación sea síncrona o asíncrona y el bit de paridad.
UDRO	Este registro conserva el dato recibido en recepción y sostiene el dato para transmisión.

Cuadro 43: Registros utilizados en el módulo UART en PIC16F887

C. Comparación módulos UART: PIC16F887 y ATmega328P

Se compara el funcionamiento de ambos módulos por medio de la cantidad de ciclos de máquina utilizados en la configuración para cada uno de los programas, tanto del microcontrolador que recibe como el que envía. No se utiliza el analizador lógico para las comunicaciones, debido que, al ser un protocolo de comunicación, debe cumplir con especificaciones de tiempos y frecuencias para poder implementarse. Por lo que la comparación se limita a los registros utilizados para realizar la comunicación y las instrucciones necesarias para llevarla a cabo.

1. Comparación cantidad de instrucciones y ciclos de reloj

Se analizan los códigos propuestos en ensamblador para implementar cada uno de los programas y se obtienen los siguientes resultados:

Instrucción	Cantidad	Ciclos	Ciclos de instrucción	Periodos de reloj
CALL	3	2	6	24
BANKSEL	7	1	7	28
GOTO	1	2	2	8
MOVLW	4	1	4	16
MOVWF	3	1	3	12
CLRF	2	1	2	8
BSF	3	1	3	12
BCF	1	1	1	4
RETURN	3	2	6	24
IORWF	1	1	1	4
Total	28			140

Cuadro 44: Conteo de instrucciones en la configuración de la recepción de datos para UART y periodos de reloj totales utilizados en PIC16F887, según programa propuesto, en ensamblador.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción	Periodos de reloj
GOTO	2	2	4	16
MOVWF	4	1	4	16
RETFIE	1	2	2	8
SWAPF	4	1	4	16
BTFSS	1	2	2	8
MOVF	1	1	1	4
Total	13			68

Cuadro 45: Conteo de instrucciones en la recepción de datos por UART utilizando interrupciones y conteo de periodos de reloj utilizados en PIC16F887, según programa propuesto, en ensamblador.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción	Periodos de reloj
CALL	3	2	6	24
GOTO	1	2	2	12
BANKSEL	9	1	9	36
MOVLW	10	1	10	40
MOVWF	9	1	9	36
IORWF	1	1	1	4
RETURN	3	2	6	24
BSF	1	1	1	4
Total	37			180

Cuadro 46: Conteo de instrucciones en la configuración de transmisión de datos por UART y periodos de reloj totales utilizados en PIC16F887, según programa propuesto, en ensamblador.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción	Periodos de reloj
GOTO	2	2	4	16
BANKSEL	2	1	2	8
MOVF	1	1	1	4
MOVWF	1	1	1	4
BTFSS	1	2	2	8
Total	7			40

Cuadro 47: Conteo de instrucciones en la transmisión de datos por UART y periodos de reloj totales utilizados en PIC16F887, según programa propuesto, en ensamblador.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción
CALL	2	4	8
RJMP	1	2	2
LDI	4	1	4
STS	4	2	8
RET	2	4	8
Total	13		30

Cuadro 48: Conteo de instrucciones en la configuración de recepción de datos en UART y ciclos de instrucción utilizados en ATmega328P, según programa propuesto, en ensamblador.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción
RJMP	2	2	2
LDS	2	1	4
SBRS	1	2	8
RETI	1	4	8
Total	6		22

Cuadro 49: Conteo de instrucciones para recepción de datos en UART utilizando interrupciones y conteo de ciclos de instrucción utilizados en ATmega328P, según programa propuesto, en ensamblador.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción
CALL	1	4	4
LDI	4	1	4
STS	4	2	8
RET	1	4	4
Total	10		20

Cuadro 50: Conteo de instrucciones para configuración de transmisión de datos en UART y ciclos de instrucción utilizados en ATmega328P, según programa propuesto, en ensamblador.

Instrucción	Cantidad	Ciclos	Ciclos de instrucción
CALL	1	4	4
LDI	4	1	4
STS	4	2	8
RET	1	4	4
LDS	1	2	2
RJMP	2	2	4
SBRS	1	3	3
Total	14		29

Cuadro 51: Conteo de instrucciones para transmisión de datos en UART y ciclos de instrucción utilizados en ATmega328P, según programa propuesto, en ensamblador.

La cantidad de ciclos de reloj utilizados para los programas propuestos en Ensamblador se presentan en el Cuadro 52, como se puede observar, la cantidad de ciclos de reloj utilizados para realizar las configuraciones de los módulos difieren en gran medida, teniendo 140 y 180 ciclos de reloj necesarios en el PIC16F887, contra los 30 y 20 utilizados en el ATmega328P. Esto se puede explicar a su vez según lo que se presenta en el Cuadro 53, en la que podemos observar que la cantidad de instrucciones necesarias en el PIC16F887 aumentan 15 para recepción y 27 para transmisión. Los ciclos de reloj necesarios para recepción tienen una menor diferencia respecto a las configuraciones, sin embargo, el ATmega328P puede completarlo de forma más rápida, debido a las configuraciones de registros R previos.

Microcontrolador	Configuración recepción	Recepción	Configuración transmisión	Transmisión
PIC16F887	140	68	180	40
ATmega328P	30	22	20	29

Cuadro 52: Comparación ciclos de reloj para UART en ensamblador.

Microcontrolador	Configuración recepción	Recepción	Configuración transmisión	Transmisión
PIC16F887	28	13	37	7
ATmega328P	13	6	10	14

Cuadro 53: Comparación de cantidad de instrucciones necesarias para configuración y ejecución del módulo UART según programas propuestos en ensamblador.

Tal como se aprecia en los cuadros 52 y 53, los ciclos de reloj utilizados en recepción son 68 utilizados en PIC, con 13 instrucciones y 22 utilizados en ATmega328P, con 6 instrucciones. En el caso de la transmisión, el PIC16F887 necesita menos instrucciones, utilizando únicamente 7 instrucciones, a diferencia del ATmega328P, el cual utiliza 14; esto cambia completamente al observar los ciclos de reloj utilizados, en los cuales el PIC16F887 utiliza 40 y el ATmega328P utiliza 29.

2. Comparación de cantidad de registros utilizados por módulo

Respecto a los registros utilizados por módulo UART, el PIC16F887 utiliza un total de 7 registros para configuración y ejecución, a diferencia del módulo UART en ATmega328P, el cual utiliza únicamente 4. Cabe mencionar que el PIC16F887 utiliza un registro para almacenar la recepción y un registro para mantener los datos por enviar. Esto no sucede en el ATmega328P, el cual utiliza únicamente un registro.

Microcontrolador	Registros
PIC16F887	BAUDCTL TXSTA RCSTA SPBRGH SPBRG RCREG TXREG
ATmega328P	UCSR0A UCSR0B UCSR0C UDR0

Cuadro 54: Comparación de cantidad de registros utilizados por módulo

Módulo de comunicación Interfaz Periférica Serial (SPI)

El estándar de comunicación SPI es utilizado en el microcontrolador PIC16F887 y en el ATmega328P. En el caso del PIC16F887, forma parte del módulo conocido como Puerto síncrono serial maestro o MSSP por sus siglas en inglés; en cambio, el ATmega328P cuenta con 2 módulos con compatibilidad para este estándar, teniendo un módulo SPI dedicado y el módulo USART en modo SPI, el último funcionando únicamente como maestro.

El estándar SPI cuenta con 4 conexiones para ejecutarse, siendo la señal de reloj, el canal de entrada, el canal de salida y la selección de chip o esclavo. El estándar cuenta con 4 modos SPI, dependiendo del estado inactivo de la señal de reloj (HIGH o LOW) y del flanco en el que se captura la señal (Flanco positivo o negativo).

Al igual que la comparación realizada en el capítulo de comunicación UART, en este capítulo se comparan ambos módulos por medio de la cantidad de registros utilizados para la configuración y operación en cada uno de los módulos, además, se realiza una comparación de envío de datos en ráfaga en ambos microcontroladores, con el objetivo de verificar si estos llegan a fallar en alguna ejecución. La comparación de datos en ráfaga se realiza utilizando el Logic Analyzer.

A. Módulo MSSP, configuración SPI en PIC16F887

El modo SPI del módulo MSSP, permite enviar y recibir simultáneamente y sincronamente 8 bits de datos. Para seleccionar el modo del módulo MSSP se deben configurar los bits SSPM [3:0] del registro SSPCON, se puede seleccionar el rol (maestro o esclavo), además, permite configurar la señal de reloj, así como si se utiliza o no la conexión SS en modo esclavo. Se presenta con más detalles cada selección en el Cuadro 55.

SSPM [3:0]	Modo de MSSP
0000	Modo SPI Maestro, Señal de reloj = $\frac{F_{osc}}{4}$
0001	Modo SPI Maestro, Señal de reloj = $\frac{F_{osc}}{16}$
0010	Modo SPI Maestro, Señal de reloj = $\frac{F_{osc}}{64}$
0011	Modo SPI Maestro, Señal de reloj = $\frac{Timer2}{2}$
0100	Modo SPI Esclavo, SS activado
0101	Modo SPI Maestro, SS funciona como pin I/O

Cuadro 55: Configuración de bits SSPM [3:0] para SPI

Como se menciona previamente, existen 4 modos de SPI. La configuración de cada uno de ellos, depende de los bits CKP y CKE, de los registros SSPCON y SSPSTAT respectivamente. La configuración y presentación gráfica del modo se presenta en la Figura 8.

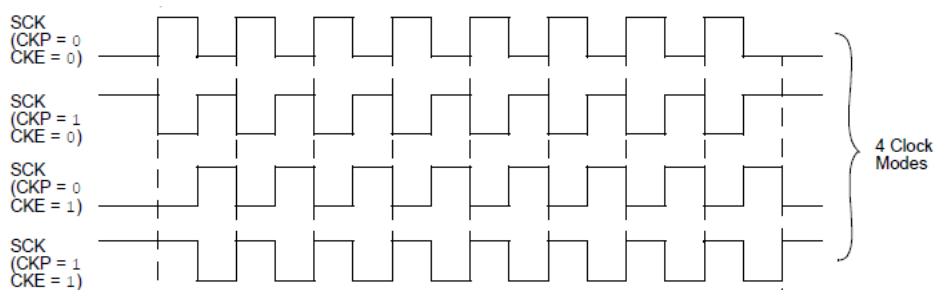


Figura 8: Modos SPI según configuración del reloj. [11]

La activación del bit SSPEN del registro SSPCON permite activar el módulo MSSP, habilitando los pines SCK, SDO, SDI y SS como parte del puerto serial; la activación de este bit no configura el registro TRIS correspondiente de cada pin, por lo que es necesario configurar los pines como entrada o salida dependiendo del rol asignado. El Cuadro 56 resume las configuraciones de los pines según el rol asignado.

Pin	Configuración	
	Maestro	Esclavo
TRISC5	SDO, se configura como salida, TRISC5 = 0.	
TRISC3	SCK, se configura como salida, TRISC3 = 0.	SCK, se configura como entrada, TRISC3 = 1.
TRISC4	SDI, se configura como entrada, TRISC4 = 1.	
TRISA5	Pin I/O, no se utiliza en este rol.	SS, se configura como entrada, TRISA5 = 1.

Cuadro 56: Configuración de registros TRISx según el rol asignado para configuración SPI en PIC16F887.

1. Comunicación por estándar SPI: Maestro

El rol de maestro en el estándar SPI, determina cuando se inicia la comunicación, en el caso del microcontrolador PIC16F887, la comunicación se inicia automáticamente cuando se coloca un dato en el registro SSPBUF; sin embargo, el esclavo solo recibirá el dato y transmitirá el propio cuando el pin SS se encuentra en 0 lógico o LOW. El bit que indica la recepción de un dato es BF del registro SSPSTAT. El proceso para llevar a cabo una comunicación se presenta en el diagrama de la Figura 9



Figura 9: Diagrama de flujo para comunicación por estándar SPI, rol maestro para PIC16F887.

2. Comunicación por estándar SPI: Esclavo

El estado del pin SS en el microcontrolador determina si el módulo MSSP está despierto o hibernando, al estar el pin SS en 1 lógico, hiberna el módulo, por lo que no consume energía; en este modo se puede seguir escribiendo al registro SSPBUF, pero la comunicación no se iniciará hasta que el módulo despierte y se inicie la señal de reloj, por medio del maestro.

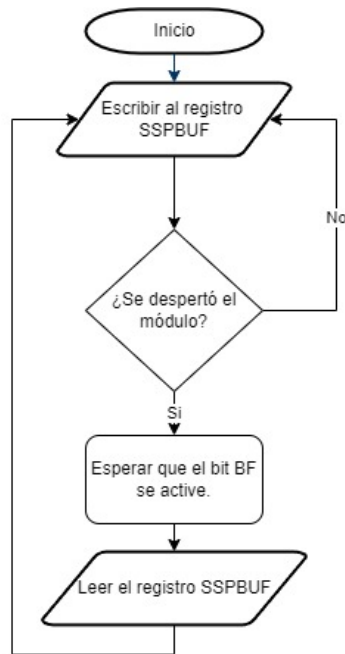


Figura 10: Diagrama de flujo para comunicación por estándar SPI, rol esclavo para PIC16F887.

3. Resumen de registros utilizados

Registro	Usos
SSPSTAT	Se utilizan únicamente los bits SMP y CKE, son utilizados para determinar el momento en que se captura el dato y momento en el que se envía o recibe el dato, respectivamente. (Determina uno de los 4 modos de SPI).
SSPCON	Activa el módulo y configura el rol del módulo. Se utiliza para determinar la polaridad del reloj.
SSPBUF	En este registro se mantiene el dato a enviar o el dato recibido (depende del papel que esté cumpliendo en la comunicación).

Cuadro 57: Registros utilizados en el módulo MSSP: configuración SPI.

B. Módulo SPI en ATmega328P

El módulo SPI del ATmega328P, permite la transferencia de 8 bits de datos de forma síncrona. El módulo se configura por medio de 9 bits, ubicados en todos en el registro SPCR exceptuando el bit SPI2X que se encuentra en el registro SPSR. El módulo SPI del ATmega328P puede funcionar bajo cualquiera de los 4 modos de SPI modificando los bits CPHA y CPOL, la configuración de los mismos se puede observar en la Figura 11,

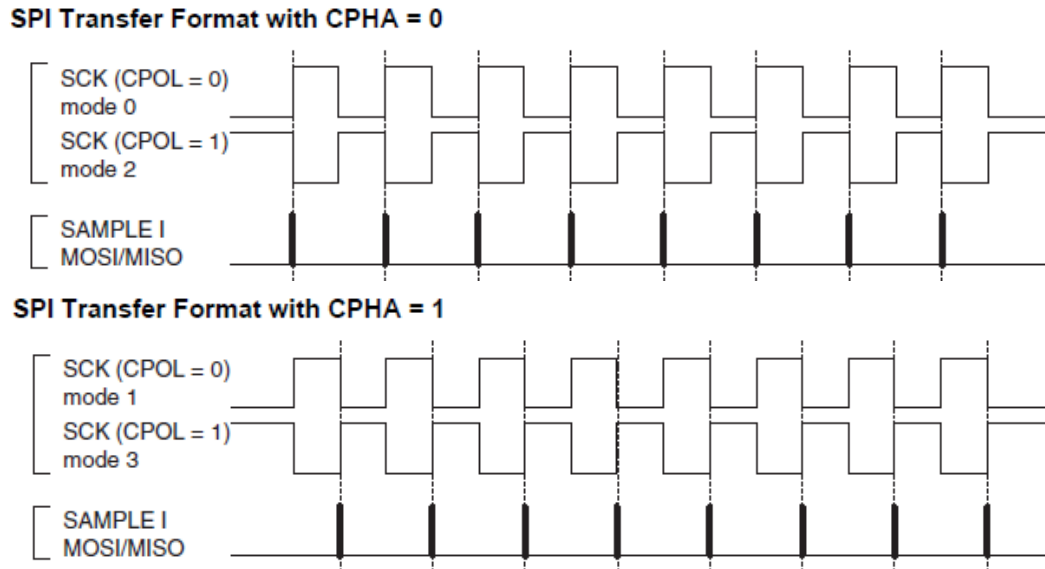


Figura 11: Modos SPI según configuración del reloj [12].

Al ser un estándar síncrono, requiere una señal de reloj para realizarse, por lo que la configuración de la señal de reloj generada por el maestro, se realiza por medio de los bits SPI2X y SPR[1:0], estas configuraciones permiten obtener una gran variedad de frecuencias para la señal de reloj a utilizar en la transmisión, tal como se presenta en el Cuadro 58.

SPI2X	SPR[1:0]	Frecuencia de la señal de reloj
0	00	$\frac{F_{osc}}{4}$
0	01	$\frac{F_{osc}}{16}$
0	10	$\frac{F_{osc}}{64}$
0	11	$\frac{F_{osc}}{128}$
1	00	$\frac{F_{osc}}{2}$
1	01	$\frac{F_{osc}}{8}$
1	10	$\frac{F_{osc}}{32}$
1	11	$\frac{F_{osc}}{64}$

Cuadro 58: Configuración de la señal de reloj para el módulo SPI de ATmega328P

Para configurar el rol del microcontrolador, se debe configurar el bit MSTR. Si se activa el bit, se configura para que sea maestro, de lo contrario, se configura como esclavo. El módulo SPI del ATmega328P, tiene la opción de modificar el orden en el que se ordenan los datos recibidos y enviados, permite configurar si el primer bit enviado y recibido es el más significativo o el menos significativo. Esta configuración se realiza por medio del bit DORD, si se activa este bit se envía primero el bit menos significativo. De lo contrario, se envía el bit más significativo de primero.

Para activar las interrupciones se debe modificar el bit SPIE, el bit SPIF del registro SPSR indica el final de una transmisión, esto dirige al vector reservado 0x0024. Por último para activar el módulo SPI se necesita activar el bit SPE. Este bit habilita todas las funciones y operaciones de la comunicación, sin embargo, no modifica el estado de los pines conectados al módulo, los cuales se deben de configurar por medio de sus respectivos registros DDRx, las configuraciones necesarias según rol establecido se presentan en el Cuadro 59.

Pin	Configuración	
	Maestro	Esclavo
DDB4	MISO, se configura como entrada, DDB4= 0.	MISO, se configura como salida, DDB4= 1.
DDB5	SCK, se configura como salida, DDB5 = 1.	SCK, se configura como entrada, DDB5 = 0.
DDB3	MOSI, se configura como salida, DDB3 = 1.	MOSI, se configura como entrada, DDB3 = 0.
DDB2	Pin I/O, no se utiliza en este rol.	SS, se configura como entrada, DDB2 =0.

Cuadro 59: Configuración de registro DDRB según el rol asignado para configuración SPI en ATmega328P

1. Comunicación por estándar SPI: Maestro

El maestro, bajo este estándar, determina cuando se realiza una transmisión, seleccionando el esclavo y posteriormente generando la señal de reloj para empezar a transmitir datos. En el caso del ATmega328P, la comunicación inicia después de escribir un dato al registro SPDR, la transmisión del dato habrá finalizado cuando la bandera SPIF se haya activado. Se presenta un diagrama que presenta el proceso para llevar a cabo la comunicación en la Figura 12.

2. Comunicación por estándar SPI: Esclavo

El módulo SPI en modo esclavo, se mantiene activo o hibernando dependiendo del estado del pin SS, mientras el mismo se encuentra en HIGH el módulo no consume energía. En este modo se puede seguir escribiendo al registro SPDR sin problemas, la transmisión solo inicia hasta que el maestro despierte el módulo por medio del pin SS e inicie la señal de reloj. El algoritmo a seguir se presenta en la Figura 13.



Figura 12: Diagrama de flujo para comunicación por estándar SPI, rol maestro para ATmega328P.

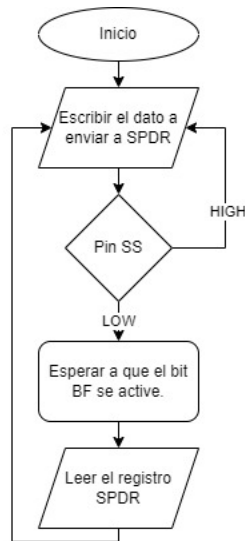


Figura 13: Diagrama de flujo para comunicación por estándar SPI, rol esclavo para ATmega328P.

3. Resumen de registros utilizados

Registro	Usos
SPCR	Es el registro de control, se configura fase y polaridad para determinar el modo de SPI, la frecuencia del reloj, el orden en el que se envían los datos, el rol y se activa el módulo y sus interrupciones.
SPSR	Se encuentra la bandera de interrupción y el bit que determina la frecuencia del reloj en conjunto con los bits SPR[1:0] del registro SPCR.
SPDR	En este registro se mantiene el dato a enviar o el dato recibido (depende del papel que esté cumpliendo en la comunicación).

Cuadro 60: Registros utilizados para el estándar SPI en ATmega328P

C. Comparación módulos MSSP modo SPI de PIC16F887 y Módulo SPI de ATmega328P

Se comparan ambos módulos por medio de la cantidad de registros que utiliza cada uno, los bits utilizados para verificar los procesos y la respuesta ante datos en ráfaga para ambos microcontroladores. Los registros utilizados para la configuración se presentan en el Cuadro 61, tal como se puede observar, el ATmega328P se configura por medio de 9 bits, con 8 de ellos en el registro SPCR. El PIC requiere 11 bits distribuidos en 4 registros.

Microcontrolador	Registros utilizados
PIC16F887	SSPCON [5:0] SSPSTAT [7:6] PIE1[3] INTCON[7:6]
ATmega328P	SPCR [7:0] SPSR [0]

Cuadro 61: Registros utilizados para la configuración y ejecución del estándar SPI en microcontroladores PIC16F887 y ATmega328P.

La verificación de transmisión en ambos microcontroladores es similar, se utiliza una bandera la cual indica el fin de la transmisión. El PIC16F887 cuenta con 2 banderas, SSPIF y BF de los registros PIR y SSPSTAT respectivamente. Ambas se activan al momento de completar la transmisión. En el ATmega328P se utiliza únicamente una bandera siendo el bit SPIF del registro SPSR. Verificar el estado de la transmisión es prácticamente el mismo en ambos microcontroladores. Tal como se puede apreciar en los siguientes ejemplos.

Verificación de estado de transmisión en PIC16F887

```
SPI_pic_Wait(){  
    while(!SSPSTATbits.BF); //Esperar que se active la bandera  
}
```

Verificación de estado de transmisión en ATmega328P

```
SPI_avr_Wait(){  
    while(!(SPSR & (1<<SPIF))); //Esperar que se active la bandera  
}
```

1. Respuesta de microcontroladores ante envío de datos en ráfaga

Se realiza un monitoreo de las señales SDI/MISO, SDO/MOSI y SCL durante el envío de datos en ráfaga para evaluar si los módulos utilizados para SPI fallan en cierto momento. El algoritmo utilizado para evaluar fallos, espera un dato y realiza una comprobación de que el dato recibido es el esperado, en caso de no ser el dato desactiva un pin para indicar que el dato recibido es erróneo. El código ejecutado selecciona el chip y permanece enviando datos en ráfaga sin desactivar en ningún momento el esclavo.

Tanto el ATmega328P como el PIC16F887 no fallaron después de ejecuciones de 14 minutos, se escoge este tiempo debido a que se utiliza un buffer de 0.5GB de datos a 25M/S de muestras. Se presenta a continuación un fragmento de los datos que se intercambian y una vista general de los datos enviados.

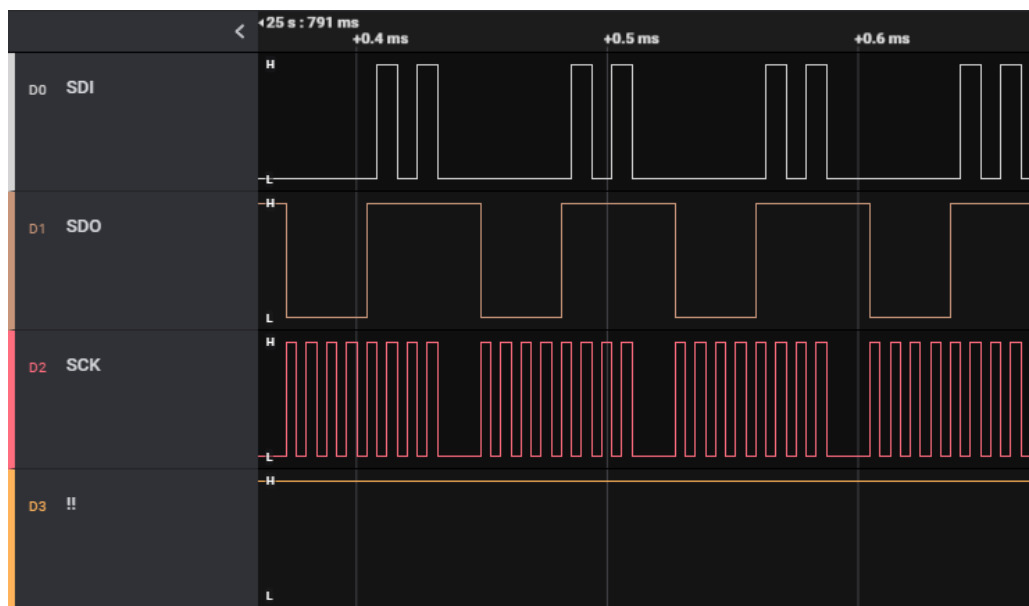


Figura 14: Fragmento de señales de canales SDI, SDO, SCK y fallo, para comunicación SPI en ATmega328P.



Figura 15: Vista completa de señal de fallo para comunicación SPI en ATmega328P.

Como se puede apreciar en la Figura 15, la señal de fallo representada con "!!", no presenta en ningún momento un flanco negativo, esto indica que el proceso de comunicación no falló en los 800 segundos de ejecución del programa. El comportamiento de las terminales del módulo a lo largo de los 800 segundos se presenta en la Figura 14.

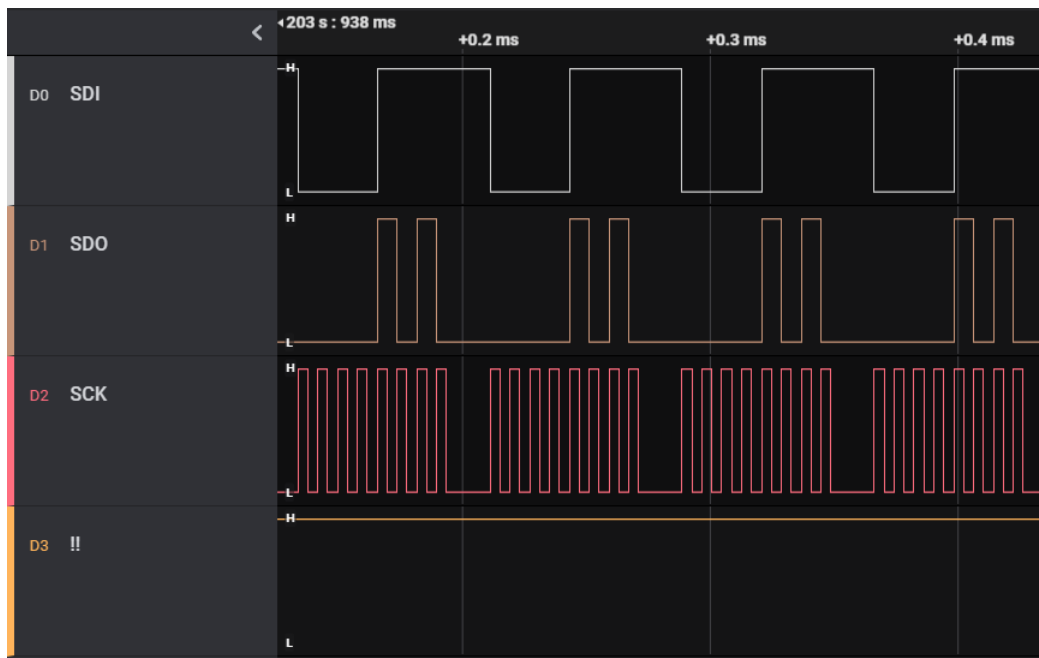


Figura 16: Fragmento de señales de canales SDI, SDO, SCK y fallo, para comunicación SPI en PIC16F887.

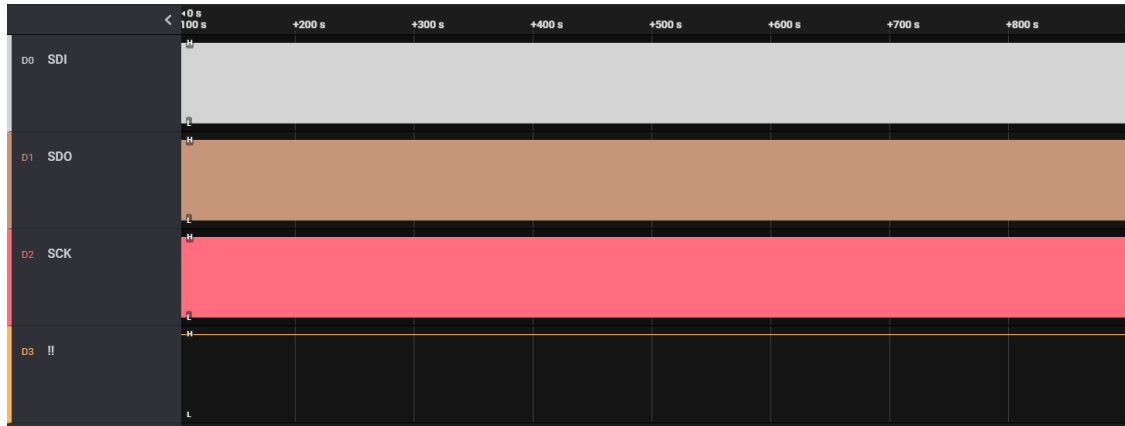


Figura 17: Vista completa de señal de fallo para comunicación SPI en PIC16F887.

Como se puede apreciar en la Figura 17, el comportamiento presentado por el PIC16F887 es el mismo que el presentado por el ATmega328P. La señal de fallo representada con !!, no presenta en ningún momento un flanco negativo, esto indica que el proceso de comunicación no falló en los 800 segundos de ejecución del programa. El comportamiento de las terminales del módulo a lo largo de los 800 segundos se presenta en la Figura 16.

Módulo de comunicación circuito inter-integrado (I2C)

El bus de datos I2C permite comunicarse de forma síncrona con varios chips por medio de 2 conexiones, siendo estas: SDA y SCL. La conexión SDA (Serial Data) permite la transmisión de datos y SCL (Serial Clock) es la señal de reloj que genera el maestro. Los pines que conforman este bus de datos son open drained por lo que para asegurar el estado inactivo se utilizan resistencias conectadas a Vcc. Esta comunicación no es *Full duplex*, por lo que no puede enviar y recibir datos al mismo tiempo. La comunicación I2C posee combinaciones de señales que determinan inicio de la transmisión, fin de transmisión e inicio repetido. Las condiciones se presentan en el Cuadro 62.

Señal	SDA	SCL
Inicio	Flanco negativo	HIGH
Parada	Flanco positivo	HIGH
Inicio repetido	Flanco negativo	HIGH

Cuadro 62: Señales de indicación para comunicación I2C

La comunicación I2C requiere, posterior a mandar una señal de inicio, enviar la dirección del esclavo al cual se quiere comunicar, además, un bit que indica si se va a escribir o leer; es importante que no hayan direcciones iguales, puesto que de lo contrario podría causar contención en el bus. Cada transmisión exitosa le sigue un bit de Acknowledge, esta señal ocurre cada 9 bit.

A. Módulo MSSP, configuración I2C en PIC16F887

El modo I2C del módulo MSSP, habilita este bus de datos. Este modo utiliza los pines RC3 Y RC4, los mismos deben estar configurados en ambos modos como entrada por medio del registro TRISC, el módulo se encarga de alternar los mismos durante la comunicación. Para seleccionar el modo I2C del módulo MSSP, se deben configurar los bits SSPM[3:0] del registro SSPCON, se puede seleccionar el rol (maestro o esclavo), además, en modo esclavo permite configurar si la dirección del esclavo es de 7 o 10 bits. La configuración se presenta en el Cuadro 63.

SSPM [3:0]	Modo de MSSP
0110	I2C modo esclavo, 7 bits de dirección
0111	I2C modo esclavo, 10 bits de dirección
1000	I2C modo maestro
1001	Cargar máscara
1010	Reservado
1011	I2C modo maestro, Firmware controlado
1100	Reservado
1101	Reservado
1110	I2C modo esclavo, 7 bits de dirección, interrupción en paro e inicio.
1111	I2C modo esclavo, 10 bits de dirección, interrupción en paro e inicio.

Cuadro 63: Configuración de bits SSPM[3:0] para I2C.

1. Comunicación I2C: Esclavo

El modo esclavo requiere definir una dirección, esta dirección será la utilizada por los demás dispositivos que quieran comunicarse con el microcontrolador. La dirección a utilizar se encuentra en el registro SSPADD.

Para configurar el módulo en modo esclavo, se debe de modificar los bits SSPM [3:0] del registro SSPCON, cabe mencionar que el modo I2C del PIC16F887 no recibe ni emite datos si alguna de las banderas WCOL y SSPOV se encuentran activas, las mismas se encuentran en el registro SSPCON y es recomendable verificar si se encuentran activas y apagarlas antes de cada comunicación. El siguiente fragmento de código en C presenta la forma adecuada de realizar esto.


```

uint8_t I2CPIC_SlaveFix(void){
    uint8_t z;
    SSPCON &= ~(1<<_SSPCON_CKP_POSITION);
    if((SSPCON & 0b01000000)|| (SSPCON & 0b10000000)){
        z = SSPBUF;
        SSPCON &= ~(1<<_SSPCON_WCOL_POSITION)&~(1<<_SSPCON_SSPOV_POSITION);
        SSPCON |= (1<<_SSPCON_CKP_POSITION);
        return z;
    }
    else{
        return 0;
    }
}

```

El código, primero mantiene bajo el reloj, al estar en *Open drained*, puede forzar el reloj para que se mantenga bajo. Esto permite pausar la comunicación mientras se verifica el estado del esclavo. Posterior a esto, el esclavo verifica el estado de los bits WCOL y SSPOV del registro SSPCON, si alguno de los dos está activo, limpia el registro SSPBUF, para luego limpiar los bits WCOL y SSPOV, por último libera la señal de reloj para que la comunicación continúe.

Escritura con esclavo

Para realizar una escritura con el esclavo, el mismo debe de haber sido llamado en un principio. Por lo que el proceso de transmisión de datos desde el esclavo inicia con la recepción de una señal *START*, esto coloca a la espera a todos los esclavos del bus. El microcontrolador debe recibir la dirección asignada previamente junto a un bit de lectura, esto le indica al esclavo que debe enviar datos al maestro. Por lo que el esclavo envía un bit de *Acknowledge* automáticamente.

Llegados a este punto, se debe de mantener bajo el reloj y se debe de cargar el dato a enviar en el registro SSPBUF. Después de cargarlo se libera el reloj y se enviarán los 8 bits de datos, recibiendo bit de *Acknowledge* cada dato, hasta que se recibe un negado, indicando por parte del maestro que ya no quiere recibir datos.

Lectura con esclavo

De la misma forma que en la escritura, la lectura inicia con una señal de *START*. Al recibir la dirección seleccionada junto con un bit de escritura, el módulo entra en modo de recepción de datos, y la dirección se almacena en el registro SSPBUF. Al recibir cada byte de datos se activa la bandera de interrupción del módulo. Enviando un bit de *Acknowledge* al finalizar cada dato.

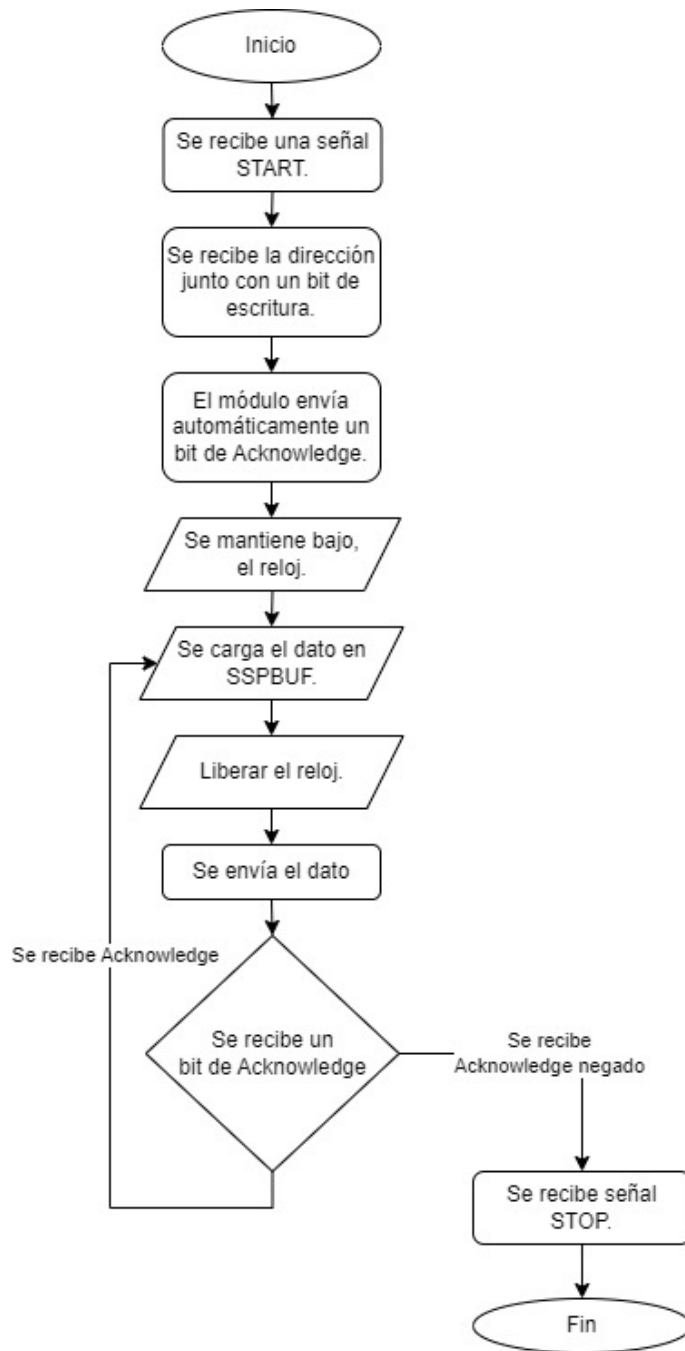


Figura 18: Diagrama de flujo para escritura en esclavo I2C en PIC16F887

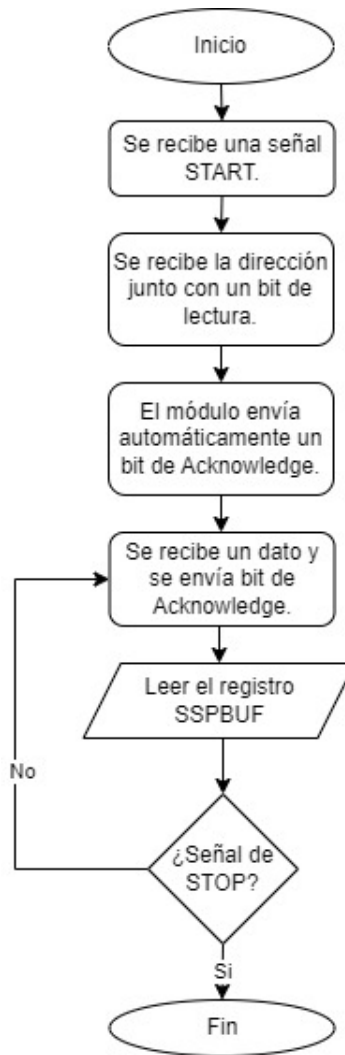


Figura 19: Diagrama de flujo para lectura en esclavo I2C en PIC16F887

2. Comunicación I2C: Maestro

La configuración inicial para maestro se consigue por medio de la modificación de los bits MSSP [3:0] del registro SSPCON, este registro logra la configuración de modo maestro con la configuración "1000" o "1011", tal como se puede apreciar en el Cuadro 63. Habiendo configurado el modo, es necesario establecer la frecuencia a la que estará trabajando el microcontrolador. Esto se realiza por medio del registro SSPADD; en modo maestro este registro se utiliza para determinar la frecuencia de la señal de reloj que se utiliza por medio de la ecuación 13

$$SSPADD = \frac{F_{osc}}{4 * Clock} - 1 \quad (13)$$

donde Clock es la frecuencia que se desea utilizar en el módulo MSSP. El módulo soporta velocidades de 100kHz, 400kHz y 1MHz. Por lo que recomiendo utilizar esos valores para determinar el valor que se utilizará en SSPADD. Al haber completado la configuración del módulo, se debe activar el bit SSPEN del registro SSPCON, el mismo activa el módulo y establece los ya mencionados pines RC3 Y RC4, en pines seriales.

Las señales utilizadas en la comunicación I2C se encuentran en el registro SSPCON2. Los bits PEN, RSEN Y SEN, corresponden a los bits de parada, inicio repetido e inicio, estos se ubican en los bits [2:0] del registro SSPCON2 respectivamente. Al momento en el que se detecta un bit de inicio, inicio repetido o parada, se activan sus correspondientes banderas del registro SSPSTAT.

Verificación acción en proceso

Para verificar que no haya ninguna transmisión en proceso, es recomendable realizar la siguiente operación:

```
void I2CPIC_Wait(void){
    while((SSPSTAT & 0x04) || (SSPCON2 & 0x1F));
}
```

Prácticamente esta operación verifica que no haya ninguna operación en proceso, por medio del bit 2 del registro SSPSTAT (R/ W), además, verifica que no hay operaciones en cola por medio de verificar que ninguno de los bits de operación esté activo, es decir los bits [4:0] del registro SSPCON2.

Señal de *Start*

Todos los inicios de una comunicación necesitan de la señal *START*, esta señal se logra, como se explica al inicio del capítulo, con un flanco negativo del canal SDA mientras el canal SCL está en HIGH. Lograr esta señal utilizando el módulo MSSP es sencillo, es necesario activar el bit SEN del registro SSPCON2. Tal como se presenta en el siguiente código:

```
void I2CPIC_Start(void){
    I2CPIC_Wait();
    SSPCON2 |= (1<< _SSPCON2_SEN_POSITION);
}
```

Como se puede observar, utilizamos la función descrita previamente para asegurarnos que no hayan acciones en cola ni transmisiones en proceso. Posteriormente activamos el bit SEN del registro SSPCON2.

Señal de *Stop*

Esta señal indica que la transmisión con el esclavo actual se quiere finalizar, esta señal requiere un flanco positivo del canal SDA mientras SCL está en HIGH. Esta señal se logra por medio del módulo MSSP activando el bit PEN del registro SSPCON2. Tal como se presenta en el siguiente código:

```
void I2CPIC_Stop(void){
    I2CPIC_Wait();
    SSPCON2 |= (1<< _SSPCON2_PEN_POSITION);
}
```

Como se puede observar, a parte de comprobar que no hay operaciones en proceso, se activa el bit PEN, iniciando la señal STOP.

Escritura con Maestro

Para escribir a un esclavo, primero se inicia la comunicación con una señal *START* y se debe de comprobar que no se está realizando ninguna otra operación en el módulo y que no hay señales en cola; además, la dirección del esclavo a escribir y el bit de escritura (7 bits de dirección, 1 de escritura/lectura), se deben de colocar en buffer SSPBUF. La dirección se habrá enviado y recibido un bit de *Acknowledge* cuando se active el bit SSPIF del registro PIR1. El proceso de escritura se replica sin la señal de *START*, colocando el dato a enviar en el buffer y esperando *Acknowledge*. El maestro finaliza la comunicación cuando envía una señal de *STOP*.

Lectura con Maestro

El proceso de lectura inicia de la misma forma que en el de escritura, el maestro debe mandar una señal de *START*, posterior a esto, mandar la dirección del esclavo con el bit de lectura/ escritura, en este caso para leer debe ser un 1. Además, requiere verificar que no hay otras operaciones en cola o en proceso. Posterior a esto se debe habilitar el modo de recepción de datos, esto se debe activar por medio del bit RCEN del registro SSPCON2. En el modo de recepción, se debe de esperar el bit SSPIF que indica que el dato ha sido recibido, el dato se puede encontrar en el registro SSPBUF. Se debe enviar un bit *Acknowledge*, esto se realiza por medio del bit ACKEN del registro SSPCON2, si el bit ACKDT del mismo registro se encuentra en 1, el *Acknowledge*, será *HIGH*, de lo contrario, será un *Acknowledge* negado, presentando un *Low*.



Figura 20: Diagrama de flujo para escritura en Maestro I2C en PIC16F887.

El *Acknowledge* determina si la comunicación continúa o se detiene, por lo que al momento de recibir datos con el maestro, se debe de enviar un *Acknowledge* negado en caso de querer finalizar la comunicación, en caso contrario no es necesario negarlo. Al recibir el último dato, se debe de enviar una señal de *STOP*. El proceso se presenta como diagrama de flujo en la Figura 21, además, se presenta un ejemplo de la implementación del código en C.

```

unsigned int I2CPIC_MasterRead(uint8_t ACK){
    uint8_t temp;
    I2CPIC_Wait();
    SSPCON2 |= (1<<_SSPCON2_RCEN_POSITION);
    I2CPIC_Wait();
    while(!SSPIF);
    temp = SSPBUF;
    I2CPIC_Wait();
    SSPCON2bits.ACKDT = ACK;
    SSPCON2 |= (1 << _SSPCON2_ACKEN_POSITION);
    return temp;
}

```

En el código se aprovecha la función "wait", explicada previamente, además, habilita la recepción y establece si el ACK a enviar está o no negado, por medio del bit ACKDT.

3. Resumen de registros utilizados

Registro	Usos
SSPSTAT	Muestra el estado de la comunicación actual, además, indica las señales recibidas.
SSPCON	Activa el módulo y configura el rol del módulo.
SSPCON2	Se utiliza para iniciar las señales de la comunicación y determinar el ACK.
SSPADD	En modo maestro determina la frecuencia de la señal de reloj del módulo. En modo esclavo es la dirección que utilizará el módulo para identificarse en el bus.
SSPBUF	En este registro se mantiene el dato a enviar o el dato recibido (depende del papel que esté cumpliendo en la comunicación).

Cuadro 64: Registros utilizados en el módulo MSSP: configuración I2C.

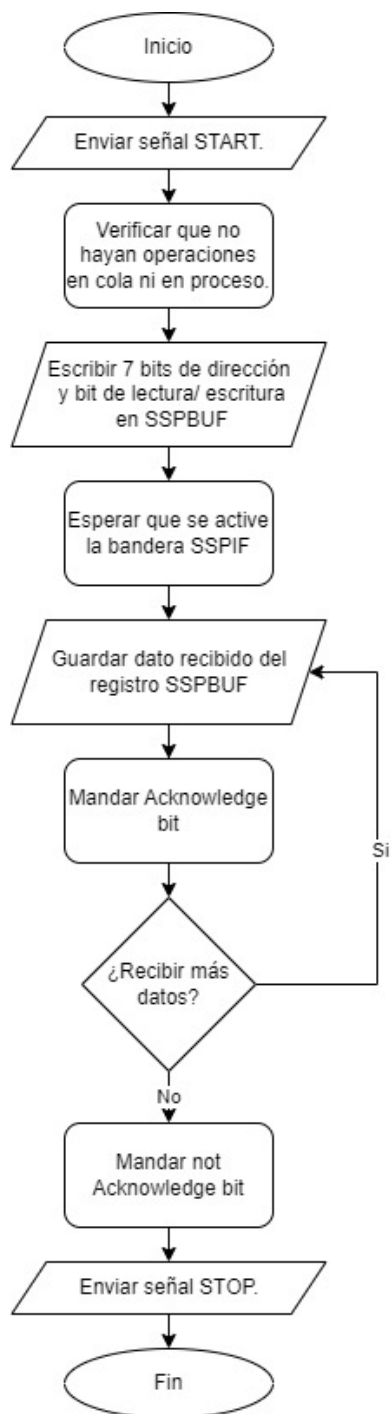


Figura 21: Diagrama de flujo para lectura en Maestro I2C en PIC16F887.

B. Módulo interfaz de 2 conexiones (TWI), compatible con I2C Philips, del ATmega328P

El módulo para comunicación por I2C de AVR se conoce como TWI, a pesar del nombre, el módulo funciona de la misma forma que el estándar I2C. El mismo se maneja a partir de un sistema que indica el estado actual de la comunicación, el código se presenta como salida del registro TWSR. El estado puede indicar el rol, si el dato que se transmite/recibe es una dirección, incluso marca si se recibe un bit *Acknowledge* o si este es negado. A continuación se presentan las configuraciones para ambos módulos.

1. Comunicación I2C: Maestro

Iniciar el módulo como maestro requiere configurar la señal de reloj que utilizará. Para esto se requiere modificar el registro TWBR en función del prescaler y frecuencia deseada utilizando la ecuación 14.

$$TWBR = \frac{\frac{F_{CPU}}{F_{SCL}} - 16}{2 * Prescaler} \quad (14)$$

Donde F_{CPU} es la frecuencia del microcontrolador y F_{SCL} es la frecuencia deseada en el módulo. Para configurar el prescaler a utilizar se deben modificar los bits TWPS[1:0] del registro TWSR. Los prescaler disponibles son 1, 4 16 y 64^1 .

Por último activar el bit TWSTA del registro TWCR, esto prepara al dispositivo para enviar una señal de Start, además de configurarlo como maestro. La señal de Stop se encuentra en el mismo registro en el bit TWSTO.

Iniciar señales

Para enviar la señal START se debe de colocar los bits TWINT, TWEN Y TWSTA, activos. Estos bits se encuentran en el registro TWCR. La señal de Start se termina cuando la bandera TWINT se encuentra apagada. Para enviar una señal STOP, se deben de activar los bits TWINT, TWEN Y TWSTO, esto ejecutará la señal de alto en la siguiente acción libre.

```
void I2CATM_Start(void){
TWCR = (1 << TWINT)|(1 << TWEN)|(1 << TWSTA);
while(!(TWCR & (1 << TWINT)));
return ;
}
```

¹En el repositorio de GitHub disponible en Anexos se incluye un Excel para determinar si el valor de TWBR será un número entero. Recomendando utilizarlo.

```

void I2CATM_Stop(void){
TWCR = (1 << TWINT)|(1 << TWEN)|(1 << TWSTO);
}

```

Se puede conocer si hubo un error en la función si el registro TWSR no indica el estado adecuado, habitualmente, mostrando el error 0x38; tal como se presenta en el Cuadro 65.

TWSR [7:3]	Estado actual del módulo
0x08	Se ha enviado una señal de START
0x10	Se ha enviado una señal repetida de START.
0x18	Se ha enviado la dirección del esclavo junto con un bit de escritura y se recibió un bit de <i>Acknowledge</i> .
0x20	Se ha enviado la dirección del esclavo junto con un bit de escritura y se recibió un bit de <i>Acknowledge</i> negado.
0x28	Se ha enviado un byte de datos y se recibió un bit de <i>Acknowledge</i> .
0x30	Se ha enviado un byte de datos y se recibió un bit de <i>Acknowledge</i> negado.
0x38	Se perdió la comunicación. Error.

Cuadro 65: Códigos de estado para maestro en transmisión.

TWSR [7:3]	Estado actual del módulo
0x08	Se ha enviado una señal de START
0x10	Se ha enviado una señal repetida de START.
0x38	Se perdió la comunicación. Error.
0x40	Se ha enviado la dirección del esclavo junto con un bit de lectura y se recibió un bit de <i>Acknowledge</i> .
0x48	Se ha enviado la dirección del esclavo junto con un bit de lectura y se recibió un bit de <i>Acknowledge</i> negado.
0x50	Se ha recibido un byte de datos y se envió un bit de <i>Acknowledge</i> .
0x58	Se ha recibido un byte de datos y se envió un bit de <i>Acknowledge</i> negado.

Cuadro 66: Códigos de estado para maestro en recepción.

Escritura con maestro

Para escribir a un esclavo, se inicia la comunicación por medio de una señal *START*, esto se verá reflejado en el maestro en el registro TWSR [7:3] como un 0x08. Se escribe la dirección del esclavo junto un bit de escritura en el registro TWDR, posterior a esto se debe de activar los bits TWINT y TWEN del registro TWCR. El estado actual pasará a ser 0x20, lo cual indica que el esclavo ha reconocido la comunicación. A partir de este momento se puede enviar datos al colocarlos en el registro TWDR y activando los bits TWINT y TWEN del registro TWCR. A partir de cada byte enviado, el estado actual que se presenta es 0x28.

Detener la comunicación con el esclavo actual, requiere enviar una señal de *STOP*, esto además, libera los canales. A continuación se presenta un diagrama que presenta el flujo que se debe de seguir para realizar una transmisión.

Lectura con maestro

El proceso de lectura inicia con una señal de *START*, este proceso se ve reflejado en el maestro con el código de estado 0x08, tal como se observa en el Cuadro 66. El proceso continúa escribiendo la dirección del esclavo junto con el bit de lectura. Se obtiene el estado actual 0x40 después que el esclavo reconoce la dirección y responde con un *Acknowledge*.

A partir de esta confirmación, se puede recibir todos los datos que se necesiten respondiendo con bit *Acknowledge* esto genera estado actual 0x50 cada vez que se recibe un dato. El último dato que se quiera recibir se debe de responder con un bit *Acknowledge* negado, de esta forma el esclavo dejará de enviar datos y el estado actual del maestro pasará a ser 0x58. Por último se debe de enviar una señal de *STOP*, de esta forma liberamos el canal y finalizamos la comunicación con el esclavo seleccionado.

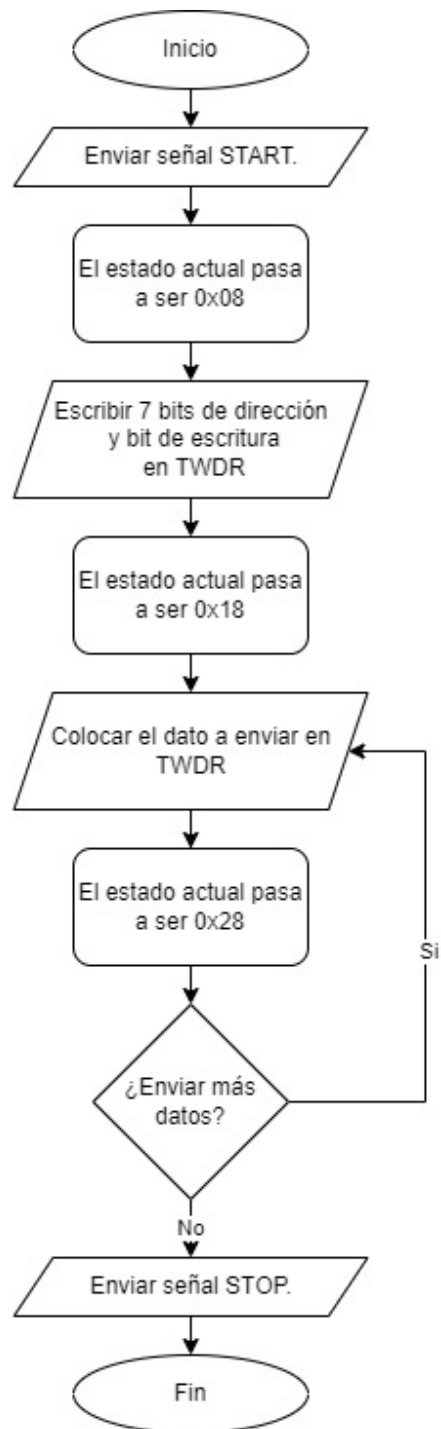


Figura 22: Diagrama de flujo para escritura en maestro I2C en ATmega328P

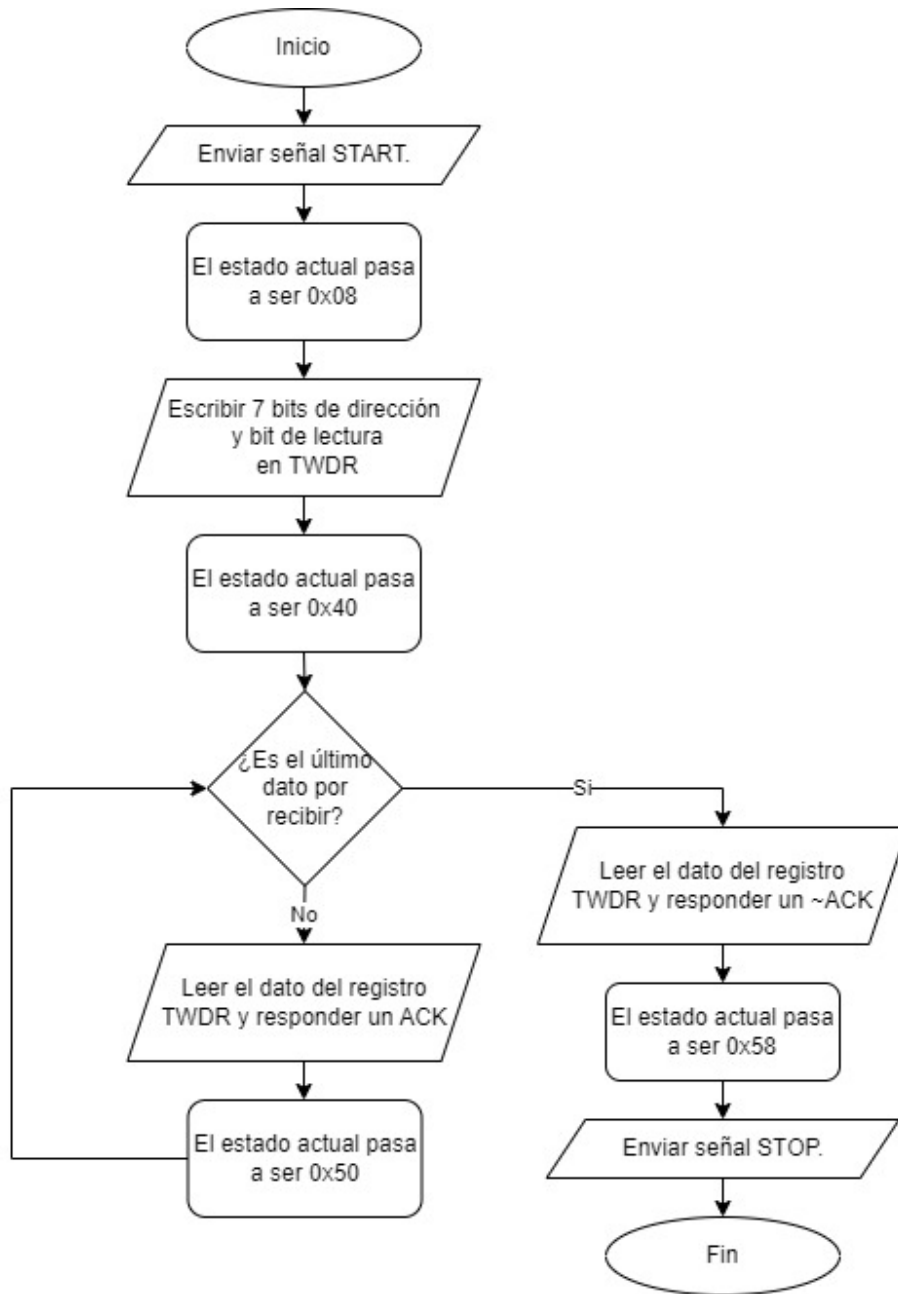


Figura 23: Diagrama de flujo para lectura en maestro I2C en ATmega328P.

2. Comunicación I2C: Esclavo

Utilizar el módulo TWI como esclavo requiere establecer una dirección con la cual identificar al dispositivo en el canal. La dirección se define por medio del registro TWAR, cabe mencionar que los 7 bits más significativos del registro funcionan como dirección, el último bit (el bit 0) funciona como habilitador de llamada general por parte del maestro.

Para configurarlo como esclavo, se deben activar los bits TWIE, TWEA, TWINT, TWEN del registro TWCR; de esta forma el módulo queda a la espera de recibir la dirección para recibir o emitir dependiendo de lo que se le solicite.

De la misma forma que en el rol de maestro, el rol de esclavo presenta diferentes códigos de estado los cuales ayudan a determinar que el proceso se está llevando de forma adecuada, los códigos de estado del rol esclavo se presentan en los cuadros 67 y 68.

TWSR	Estado actual del módulo
0x60	Se ha recibido la dirección propia, junto con bit de escritura y se envió un bit de <i>Acknowledge</i> .
0x68	
0x70	Se recibió 0x00 en datos y se envió un bit de <i>Acknowledge</i> .
0x78	
0x80	Habiendo recibido previamente un código 0x60 o 0x68, se ha recibido un dato y se envió un bit de <i>Acknowledge</i> .
0x88	Habiendo recibido previamente un código 0x60 o 0x68, se ha recibido un dato y se envió un bit de <i>Acknowledge</i> negado.
0x90	Habiendo recibido un código 0x70 o 0x78, se ha recibido un dato y se envió un bit de <i>Acknowledge</i> .
0x98	Habiendo recibido un código 0x70 o 0x78, se ha recibido un dato y se envió un bit de <i>Acknowledge</i> negado.
0xA0	Se recibió una señal de STOP o START repetido mientras se está en modo de escritura por parte del maestro.

Cuadro 67: Códigos de estado para esclavo en recepción.

TWSR	Estado actual del módulo
0xA8	Se ha recibido la dirección propia junto con bit de lectura (transmisión esclavo) y se envió un bit de <i>Acknowledge</i> .
0xB0	
0xB8	Se transmitió un byte de datos del registro TWDR y se recibió un bit de <i>Acknowledge</i> .
0xC0	Se transmitió un byte de datos del registro TWDR y se recibió un bit de <i>Acknowledge</i> negado.
0xC8	El último byte del registro TWDR ya ha sido enviado, se ha recibido un bit de <i>Acknowledge</i> .

Cuadro 68: Códigos de estado para esclavo en transmisión.

Lectura con esclavo

Es necesario configurar los bits mencionados previamente en el capítulo, esto permite al esclavo quedar a la espera de recibir su dirección y un bit de lectura/escritura. En este caso necesitamos que un bit de escritura (por parte del maestro) se reciba. El módulo activa el bit de bandera TWINT y envía un bit de *Acknowledge* inmediatamente al recibir su dirección y el bit de escritura. A este punto el código de estado actual es 0x60.

A partir de este estado, el módulo puede recibir todos los datos necesarios. Siguiendo un proceso de recepción de byte de datos y enviando bit de *Acknowledge*, media vez la comunicación sea exitosa. Esto representará un código de estado 0x80.

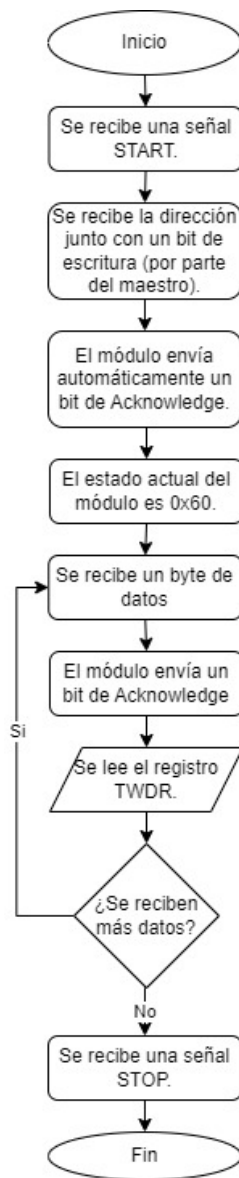


Figura 24: Diagrama de flujo para lectura en esclavo I2C en ATmega328P.

Escritura con esclavo

De la misma forma que en la lectura con esclavo, se completa la configuración del esclavo y al momento de recibir la dirección junto un bit de lectura (por parte del maestro), el módulo activa la bandera TWINT y puede empezar a enviarse datos que se encuentren en el registro TWSR. El código de estado a este punto es 0xA8. El esclavo debe verificar que se estén recibiendo bits de *Acknowledge* al final de cada transmisión de un byte de datos, es decir, verificar que el estado sea equivalente a 0xB8.

Al momento en el que se recibe un bit de *Acknowledge* negado, el maestro indica que ya no se recibirán datos. El estado del módulo será 0xC0 y en caso ya no hayan datos por enviar y el maestro siga solicitando que el esclavo transmita, el código de estado por parte del esclavo será equivalente a 0xC8.

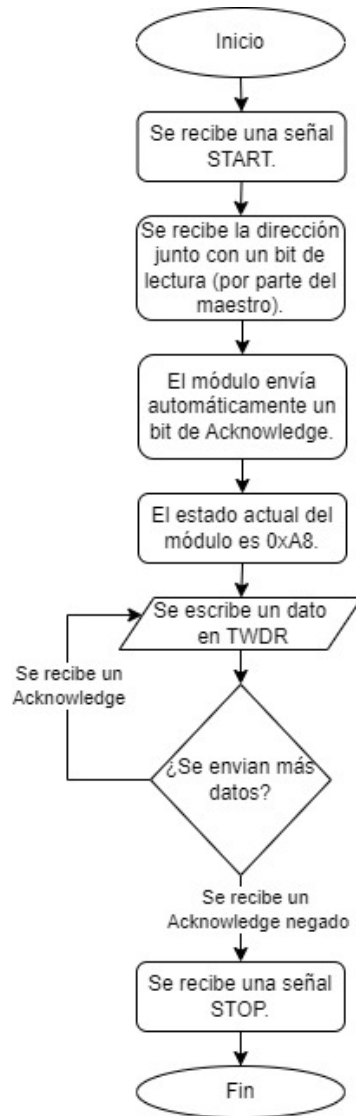


Figura 25: Diagrama de flujo para escritura en esclavo I2C en ATmega328P.

3. Resumen de registros utilizados

Registro	Usos
TWBR	Este registro cumple la función de modificar la frecuencia del reloj del módulo.
TWCR	Este registro se utiliza para controlar el módulo, permite activar el módulo, interrupciones, recepción, transmisión o respuesta automática de bit de Acknowledge.
TWSR	Este registro presenta el estado del módulo, la combinación de los bits [7:3] indican la operación que se llevó a cabo. Los bits [1:0] permiten seleccionar el prescaler para el reloj.
TWDR	En este registro se cargan los datos a enviar, así como los datos que se reciben se almacenan en el mismo.
TWAR	Este registro presenta la dirección en modo esclavo, en modo maestro este registro no se utiliza.

Cuadro 69: Registros utilizados en el módulo TWI (I2C)

C. Comparación módulos MSSP modo I2C del PIC16F887 y Módulo TWI compatible con I2C del ATmega328P

De la misma forma que se compara el módulo SPI, comparamos los módulos compatibles con I2C por medio de los registros que los mismos ocupan, además, de evaluar la respuesta de los módulos ante el envío de datos en ráfaga. Los registros utilizados por el PIC16F887 son un total de 5 registros para configuración y ejecución. De la misma forma, el ATmega328P utiliza 5 registros, de estos únicamente 4 son utilizados en modo maestro. Se puede ver la comparación en el Cuadro 70.

Microcontrolador	Registros
PIC16F887	SSPSTAT SSPCON SSPCON2 SSPADD SSPBUF
ATmega328P	TWBR TWCR TWSR TWDR TWAR

Cuadro 70: Registros utilizados para la configuración y ejecución de la comunicación I2C en microcontroladores PIC16F887 y ATmega328P.

1. Respuesta de microcontroladores ante envío de datos en ráfaga

Se realiza un monitoreo a las señales SCK Y SDA durante el envío de datos en ráfaga para verificar si los módulos utilizados para I2C (TWI en ATmega328P) fallan en cierto momento. Los análisis se realizan mediante el uso del Logic Analyzer con un buffer máximo de 0.5GB, los monitoreos de ambos microcontroladores se ejecutan por 600 segundos a 25MS/s. El algoritmo utilizado para evaluar fallos, espera un dato y realiza una comprobación de que el dato recibido es el esperado, en caso de no ser el dato, activa un pin para indicar que el dato recibido es erróneo. En el proceso de comunicación para realizar el mismo, se envía una señal de inicio, la dirección, el dato y una señal de paro; este proceso se repite de forma indefinida, el esclavo al recibir el dato lo compara contra el dato esperado.

El PIC16F887 falla después de 448 segundos de ejecución, presentando un dato erróneo en recepción. Esto se puede observar en la Figura27 y el error se puede observar en la Figura 26. El ATmega328P no presentó fallos en el envío de datos en ráfaga en los 530 segundos ejecutados, el tiempo de grabación fue inferior debido al buffer de 0.5GB. La ejecución completa se presenta en la Figura 29 y un fragmento de las señales se presentan en la Figura 28.

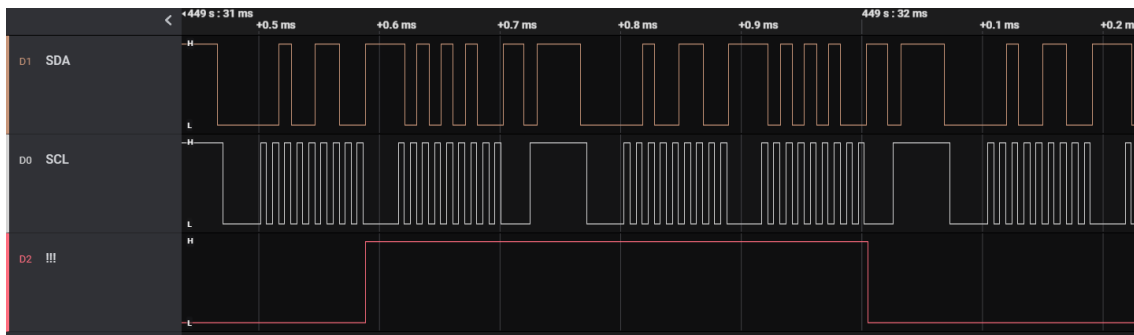


Figura 26: Fragmento de señales en error, para canales SCK, SDA y fallo, comunicación I2C en PIC16F887.



Figura 27: Vista completa de señal de fallo para comunicación I2C en PIC16F887.

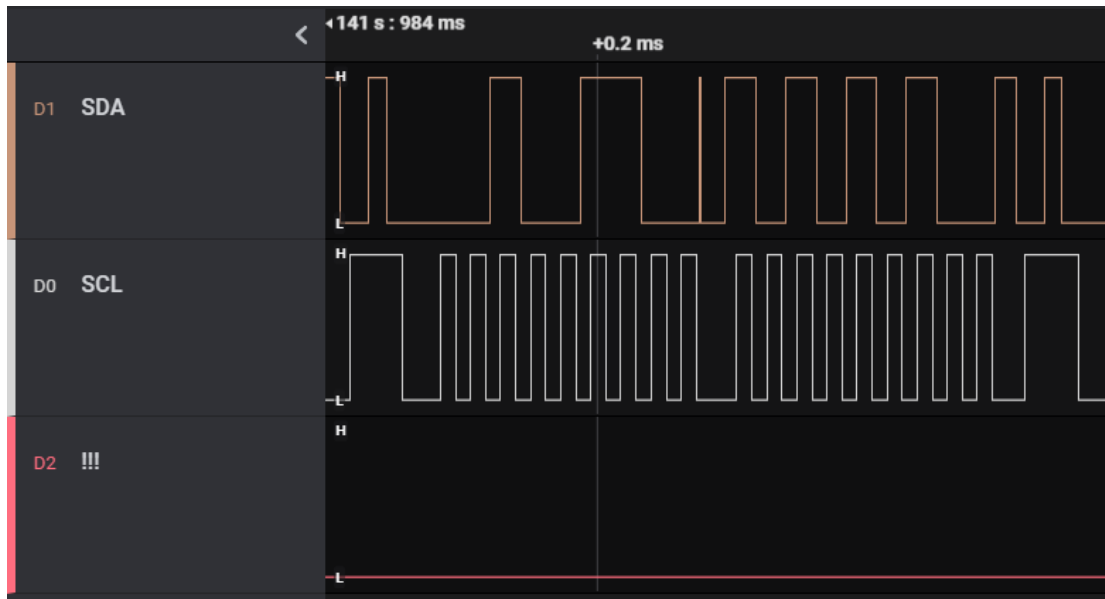


Figura 28: Fragmento de señales en canales SCK, SDA y fallo para comunicación I2C en PIC16F887.



Figura 29: Vista completa de señal de fallo para comunicación I2C en ATmega328P.

El módulo ADC del ATmega328P es más eficiente que el módulo ADC del PIC16F887. Los tiempos de resolución del ATmega328P son inferiores a los del PIC16F887 por $12\mu s$ en C, no se logra evidenciar la eficiencia en ensamblador debido al código ineficiente realizado, sin embargo, la eficiencia del ATmega328P contra el PIC16F887 se puede notar también en el tiempo entre lecturas, obteniendo $5\mu s$ menos en ensamblador y $18\mu s$ en C. Además, el ATmega328P es más eficiente para la ejecución de códigos, tal como se presentó en el capítulo 7, la cantidad de ciclos de máquina necesarios para realizar la configuración y ejecución del programa, es menor en el ATmega328P; con 47 ciclos de máquina en la configuración y 70 en la ejecución. Contra 192 de configuración y 180 de ejecución en el PIC16F887. Esto se debe analizar en conjunto con la cantidad de instrucciones necesarias utilizadas, el PIC16F887 requiere 39 instrucciones para configuración y 37 para ejecución según el código en ensamblador propuesto. El ATmega328P utiliza 27 en configuración y 35 en ejecución. Respecto a los registros utilizados por cada microcontrolador, el PIC16F887 utiliza 4 registros y el ATmega328P utiliza 2.

El módulo PWM del PIC16F887 es más preciso en obtener una frecuencia deseada, respecto al ATmega328P en modo de FAST PWM. En el cual se buscaba obtener una frecuencia de 488.28125Hz, el cual es una frecuencia estándar en el ATmega328P; en el mismo consigue 476 Hz con un error de 2.45 % y el PIC16F887 logra una frecuencia de 487.628 Hz, con un error de 0.13 %. Respecto a la cantidad de ciclos de reloj se utilizan 104 en configuración y 8 para ejecución, en cambio en ATmega328P se utilizan 24 ciclos para configuración y 2 para ejecución. Se presenta una diferencia de 80 ciclos de reloj en configuración y 6 para ejecución. Respecto a la cantidad de instrucciones utilizadas el PIC utiliza 21 instrucciones para configuración y 2 para ejecución, mientras que el ATmega328P utiliza 12 para configuración y 2 para ejecución. El ATmega328P es más eficiente ejecutándose respecto al PIC16F887 en este módulo, utilizando menos instrucciones y menor cantidad de ciclos de reloj. Ambos microcontroladores utilizan la misma cantidad de registros para configurar el módulo.

El módulo UART del ATmega328P es el módulo con menor cantidad de ciclos de reloj utilizados para configuración y ejecución de los modos transmisión y recepción. El mismo requiere en cuanto a configuración: 30 en recepción y 20 en transmisión, respecto a ejecución: 22 de recepción y 29 de transmisión. El PIC16F887 utiliza en configuración: 140 en recepción y 180 en transmisión. Para ejecución: 68 de recepción y 40 de transmisión. El ATmega328P es a su vez, el microcontrolador que utilizó menos instrucciones en 3 de 4 situaciones, teniendo menos instrucciones utilizadas en configuraciones de transmisión y recepción, así como ejecutar la recepción. El PIC16F887 utiliza menos instrucciones en el proceso de transmisión, utilizando la mitad de las instrucciones requeridas por el ATmega328P, las cantidades utilizadas por los mismos son 14 por el ATmega328P y 7 por el PIC16F887. El ATmega328P utiliza menos registros que el PIC16F887, utilizando únicamente 4 registros en lugar de los 7 utilizados por el PIC.

Respecto a los módulos de comunicación MSSP modo SPI del PIC16F887 y el SPI del ATmega328P. Se compararon la cantidad de registros utilizados por cada uno, en el cual el PIC16F887 cuenta con 4 registros contra los 2 utilizados por el ATmega328P. Se evaluó la respuesta al envío de datos en ráfaga por parte de ambos microcontroladores, ningún microcontrolador presentó errores en las ejecuciones de 800 segundos.

Por último, el módulo MSSP modo I2C del PIC16F887 utiliza la misma cantidad de registros para su configuración, que el módulo Interfaz de dos cables (TWI) compatible con I2C del ATmega328P. El envío de datos en ráfaga del PIC16F887 presentó un error en la recepción de datos después de 448 segundos, a diferencia del ATmega328P, el cual no presentó fallos en la comunicación en los 530 segundos de ejecución.

- No existe documentación con respecto a la resolución exacta del ATmega328P a diferentes frecuencias en modo de corrida libre del módulo ADC, por lo que se recomienda hacer pruebas para conocer la resolución exacta con frecuencias mayores a $200kHz$.
- Se recomienda revisar las configuraciones que se colocan en los FUSES del ATmega328P, estas características influyen en gran medida al comportamiento del microcontrolador y los voltajes a los que funciona.
- En comunicación UART para el ATmega328P, se puede utilizar la bandera de data register empty para recibir 2 datos continuos, sin embargo, esto puede causar inconvenientes para solo leer un dato. Se recomienda tener esto en cuenta.

-
- [1] D. de Electrónica Mecatrónica y Biomédica de la Universidad del Valle de Guatemala., *Programación de microcontroladores*, 2019.
 - [2] G. T. UNIVERSITY, *COURSE CURRICULUM COURSE TITLE: PIC MICRO-CONTROLLER AND EMBEDDED SYSTEMS*, Último acceso el 29 de abril de 2022, 2022. dirección: https://www.gtu.ac.in/syllabus/NEW_Diploma/Sem6/3361705.pdf.
 - [3] M. I. of Technology, *PERFORMANCE ENGINEERING OF SOFTWARE SYSTEMS*, Último acceso el 29 de abril de 2022, 2022. dirección: <https://ocw.mit.edu/courses/6-172-performance-engineering-of-software-systems-fall-2018/pages/syllabus/>.
 - [4] L. Fried, *PIC vs. AVR*, Último acceso el 29 de abril de 2022, 2004. dirección: <http://www.ladyada.net/library/picvsavr.html>.
 - [5] B. Nutter, *Microcontrollers*, Último acceso el 29 de abril de 2022, 2020. dirección: <https://www.depts.ttu.edu/ece/undergrad/syllabi/documents/ECE3362.pdf>.
 - [6] B. Nutter, *Microprocessor Architecture*, Último acceso el 30 de abril de 2022, 2020. dirección: <https://www.depts.ttu.edu/ece/undergrad/syllabi/documents/ECE4375.pdf>.
 - [7] D. Taylor, *Microcomputers*, Último acceso el 30 de abril de 2022, 2020. dirección: <https://oiraweb.ua.edu/apis/docs/api/v1/renderDocument/id/5e0fa44f5c44c672d37d21f6?contextId=20201011942>.
 - [8] D. de Electrónica Mecatrónica y Biomédica de la Universidad del Valle de Guatemala., *Electrónica digital 2*, 2020.
 - [9] M. Palmer, *A Comparison of 8-Bit Microcontrollers*, Último acceso el 7 de junio de 2022, 1997. dirección: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.468.5049&rep=rep1&type=pdf>.
 - [10] J. Sanchez y M. P. Canton, *Microcontroller Programming: The Microchip PIC®*. CRC press, 2018.

- [11] Microchip, *PIC16F882/883/884/886/887 Data Sheet*, Último acceso el 7 de junio de 2022, 2015. dirección: <https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/DataSheets/40001291H.pdf>.
- [12] Microchip y Atmel, *megaAVR® Data Sheet*, Último acceso el 7 de junio de 2022, 2020. dirección: <https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/ATmega48A-PA-88A-PA-168A-PA-328-PDS-DS40002061B.pdf>.
- [13] M. Barr, “Pulse width modulation,” *Embedded Systems Programming*, vol. 14, n.º 10, págs. 103-104, 2001.
- [14] Y.-y. Fang y X.-j. Chen, “Design and simulation of UART serial communication module based on VHDL,” en *2011 3rd International Workshop on Intelligent Systems and Applications*, IEEE, 2011, págs. 1-4.
- [15] F. Leens, “An introduction to I²C and SPI protocols,” *IEEE Instrumentation & Measurement Magazine*, vol. 12, n.º 1, págs. 8-13, 2009.
- [16] A. M. Romanov, “An easy to implement logic analyzer for long-term precise measurements,” *HardwareX*, vol. 9, e00164, 2021.
- [17] W. M. Waite y G. Goos, *Compiler construction*. Springer Science & Business Media, 2012.
- [18] R. L. Britton, *MIPS assembly language programming*. Pearson/Prentice Hall, 2004.
- [19] P. Santamaría Santiago y Valarezo, *Diseño y construcción de un módulo de comunicación Half-Duplex de datos, para los puertos serial y paralelo de un computador, asistido por un programa computacional para el monitoreo y control de la comunicación*, Último acceso el 26 de diciembre de 2022, 2003. dirección: <https://bibdigital.epn.edu.ec/bitstream/15000/11468/1/T2171.pdf>.

Repositorio de códigos y capturas de analizador lógico

https://github.com/Efajardo18/EstudioComparativo_AVRvsPic

- Acknowledge:** Traducido literalmente como reconocimiento, es una señal que indica que se ha recibido correctamente los datos. Es decir, un acuse de recibo [19].. 67
- baudaje:** Se refiere al número de símbolos que se pueden enviar por segundo, un símbolo puede tener 1 o más bits, por lo que no hay que confundirlo con tasas de datos.. 43
- DIP:** *Dual In-line Package* o empaquetado de doble hilera, es el encapsulamiento para integrados, el cuál contiene 2 hileras de pines.. 23
- duplex:** Un sistema duplex, también conocido como Full Duplex, permite la comunicación en ambos sentidos, sin importar si el otro dispositivo está transmitiendo o recibiendo en cualquier instante. Este sistema implica 2 canales separados en los que se envía y recibe respectivamente [19].. 14
- HIGH:** Se refiere al estado lógico encendido, en programación un 1 lógico y en el área digital se refiere a un voltaje que se encuentra entre el umbral de encendido.. 13
- LOW:** Se refiere al estado lógico apagado, en programación un 0 lógico y en el área digital se refiere a un voltaje que se encuentra entre el umbral de apagado.. 13
- lsb:** Se refiere al bit menos significativo de una palabra, en otras palabras el bit más a la derecha. En una palabra de n bits, el lsb es el bit 0.. 18
- msb:** Se refiere al bit más significativo de una palabra, en otras palabras el bit más a la izquierda. En una palabra de n bits, el msb es el bit n-1.. 18
- multiplexor:** Un multiplexor o MUX, es un circuito combinacional de una varias entradas y una sola salida. Este permite seleccionar una de varias entradas de datos en función de la combinación de sus entradas de control.. 17
- open drained:** Este tipo de terminal en componentes CMOS implica que se pueden conectar 2 salidas por lo que si una de las 2 es baja, la otra también se comportará de esta forma [19].. 67

Semiduplex: Un sistema semiduplex, también conocido como half duplex, es aquel que involucra una comunicación bidireccional, sin embargo, transmitiendo uno a la vez [19].. 14

Simplex: Un sistema Simplex es diseñado para transmitir datos en una sola dirección, lo cual implica que la comunicación se limita a un transmisor y un receptor permanentes [19].. 14