UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

TESIS

SISTEMA ELECTRÓNICO DE COMUNICACIÓN PACIENTE-PERSONAL MÉDICO, UTILIZANDO EL BUS I2C

QUE PARA OBTENER EL TITULO DE INGENIERO

ELÉCTRICO- ELECTRÓNICO

PRESENTA

MARIO ALBERTO FLORES HERNÁNDEZ

TUTOR

M.I. ROBERTO TOVAR MEDINA

2015

CONTENIDO

CAPITULO 1: INTRODUCCIÓN 1

1.1 Funcionalidad de un sistema de comunicación paciente- personal de enfermería dentro de un hospital 1

1.2. Cuadro comparativo de buses seriales (RS232, SPI, CAN, LIN, RS485). 3

1.3 ¿POR QUÉ I2C? 7

CAPITULO 2: BUS I2C 8

2.1 EL BUS I2C 8

2.2 Hardware del bus I2C 9

2.3 Transferencia de un bit por la línea SDA 11

2.4 Condiciones de start y stop 11

2.5 Transferencia de datos 12

2.6 Formato de transferencia de datos 13

2.7 Tipos de formatos de transferencia 14

2.8 Conexión del bus I2Centre PIC 16f628A y PIC16f886 16

CAPITULO 3: PROGRAMACION DEL PIC 17

3.1 Funciones básicas de PIC-C para la gestión del protocolo I2C 17

3.1.1 I2C_start(); 19

3.1.2 I2C_write(); 20

3.1.3 I2C_stop(); 20

3.1.4I2C_read(); 21

3.1.5I2C_poll(); 21

3.2 Diagramas de flujo: maestro y esclavo 23

3.3 Código fuente del maestro 25

3.4 Código fuente del esclavo 30

CAPITULO 4: TRANSMISIÓN REMOTA 33

4.1. introducción 33

4.2. C.I.MAX 232 33

4.3. Pruebas 35

CAPITULO 5: IMPLEMENTACION DE LATCHES 45

5.1 Introducción 45

5.2 Arquitectura del sistema 47

5.2.1. Sistema lógico combinacional 47

5.2.2. Sistema lógico secuencial 50

5.3 Memoria EEPROM interna del PIC 60
5.3.1 Introducción 60
5.3.2 Estructura interna del Microcontrolador 61
5.3.3 Memoria de programa 62
5.3.4 Contador de programa 63
5.3.5 Memoria de datos 65
5.3.6 PIC-C en la programación de la EEPROM interna del PIC16F886A 70
5.3.7 Código fuente para el control de los LATCH. 71
CAPITULO 6: PRUEBAS Y CONCLUSIONES 77
6.1 implementación física 77
6.2 Resultados 78
BIBLIOGRAFÍA 80
APÉNDICE

INDICE DE ILUSTRACIONES

CAPITULO 1: INTRODUCCIÓN.

Fig. 1.1.a Módulo de pacientes del Hospital "Venados" del IMSS 1

Fig. 1.1.b Módulo electrónico en el control de enfermería del Hospital de "Venados" del IMSS.....2

Fig. 1.1.c Diagrama de comunicación entre módulos 2

Fig. 1.2.a Comunicación SPI 4

Fig. 1.2.b Comunicación RS-485 4

Fig. 1.2.c Comunicación CAN 5

Fig. 1.2.d FamiliasPICs 6

Fig. 1.2.e Tabla de comparación de protocolos 6

CAPITULO 2: BUS I2C

Fig. 2.1.a Estructura de BUS I2C 8

Fig. 2.2.a Etapa de salida de los dispositivos I2C 10

Fig. 2.2.b Capacitancia entre líneas 10

Fig 2.3.a. Transferencia por la línea SDA 11

Fig 2.4.a. Condiciones de StaryStop 11

Fig 2.5.a Transferencia de datos 12

Fig2.5.b Bit de reconocimiento ACK 13

Fig 2.6.a Transferencia completa de datos 13

Fig 2.6.b Direccionamiento del esclavo 14

Fig 2.7.a Transferencia maestro-emisor al esclavo-receptor 15

Fig 2.7.b Transferencia maestro-receptor al esclavo-emisor 15

Fig 2.8.a Diagrama de conexión entre dispositivos maestros- emisor al esclavo-receptor 16

CAPITULO 3: PROGRAMACION DEL PIC

Fig. 3.3.a Debounce de un interruptor 27

Fig. 3.3.b Conexión entre dip-switch y PIC16f819 28

Fig. 3.3.c Conexión dispositivos de salida y entrada en puerto A 29

Fig. 3.4.a Cuadro comparativo entre distintos PICs de la misma familia 31

CAPITULO 4: TRANSMISIÓN REMOTA

Fig. 4.2.a Conexión del MAX232 con los correspondientes capacitores

Multiplicadores de voltaje 33

Fig. 4.2.b Diagrama de conexiones entre dispositivos esclavo y maestro..... 34

Fig. 4.3.a Señal SDA al salir delPIC16f819 35

Fig. 4.3.b Señal SCL al salir del PIC16f819..... 36

Fig. 4.3.c Señal SDA en nivel de voltaje RS232 36

Fig. 4.3.d Señal SCL en nivel de voltaje RS232 37

Fig. 4.3.e Señal SDA del PIC819, prueba con 100 metros de cable telefónico sin el MAX232.. 36

Fig. 4.3.f Señales SDA y SCL al salir del PIC16f819 38

Fig. 4.3.g Señales SDA y SCL al salir del MAX232delcircuito maestro 39

Fig. 4.3.hDiagrama del circuito equivalente para una conexión I2C 40

Fig. 4.3.iImplementación de filtros paso-bajas y seguidores de voltaje con el TL082 a la entrada del dispositivo esclavo..... 41

Fig. 4.3.j Diseño de filtro paso-bajas de primer orden para una frecuencia

de corte de 100 Khz 41

Fig. 4.3.k Diseño de filtro paso-bajas de primer orden para una frecuencia

de corte de 120 Khz 42

Fig. 4.3.lSeñal SDA al pasar por el filtro paso-bajas 42

Fig. 4.3.mSeñal SCL al pasar por el filtro paso-bajas 43

Fig. 4.3.n Señales SDA y SCL que llegan al dispositivo esclavo .....:..... .43

Fig. 4.3.o Comparación de las señales SCL del esclavo (1) y del maestro (2)..... 44

Fig. 4.3.p Comparación de las señales SDA del esclavo (1) y del maestro (2)..... 44

CAPITULO 5: IMPLEMENTACION DE LATCHES

Fig. 5.1.a Prototipo módulo de control de enfermería con indicadores led 45

Fig. 5.1.b Diagrama de conexión entre el PIC16f886 y los latches 46

Fig. 5.2.a Representación de un sistema combinacional 47

Fig. 5.2.b Compuerta AND 48

Fig. 5.2.c Compuerta OR.....48

Fig. 5.2.d Compuerta NOT 48

Fig. 5.2.e Compuerta YES 48

Fig. 5.2.f Compuerta NAND 49

Fig. 5.2.g Compuerta NOR 49

Fig. 5.2.h Modelo de circuito secuencial 51

Fig. 5.2.i Biestable elemental 51

Fig. 5.2.j Biestable elemental con las terminales de salida Q y Q' 51

Fig. 5.2.k Biestable elemental con valores para Q y Q' 52

Fig. 5.2.l Biestable S-R 52

Fig. 5.2.mBiestable S-R con valores para S=0 y R=0 52

Fig. 5.2.n Biestable S-R con R =0, S=1 y Q=0 como estado inicial 53

Fig. 5.2.o Biestable S-R con R =0, S=1 y Q=1 como estado inicial 53

Fig. 5.2.p R=S=1 combinación prohibida 53

Fig. 5.2.q Tabla de transiciones 54

Fig. 5.2.r Tipos de sincronización 55

Fig. 5.2.s Biestable S-C sincronizado por nivel 1..... 56

Fig. 5.2.t Símbolo y tabla de transiciones del biestable S-C 56

Fig. 5.2.u Biestable S-C sincronizado por flanco ascendente 57

Fig. 5.2.v Biestable j-k sincronizado por flanco ascendente con entradas asíncronas 57

Fig. 5.2.w Biestable D sincronizado por nivel 1.....58

Fig. 5.2.x Biestable D sincronizado por flanco ascendente 58

Fig. 5.2.aa Integrado 74LS573N ……59

Fig. 5.2.bb Tabla de verdad 74LS573N ……60

Fig. 5.2.cc Diagrama lógico latch octal 74LS573N …… 60

Fig. 5.3.a Arquitectura Von Neumann ……61

Fig. 5.3.b Arquitectura Harvard ……62

Fig. 5.3.c Mapa de memoria de programa del PIC16f886 ……62

Fig. 5.3.d Diagrama de bloques del PIC16f886 ……64

Fig. 5.3.e Tamaño de la memoria de datos del PIC16f886 ……65

Fig. 5.3.f Configuración de RP1:RP2 para la selección del banco a utilizar ……66

Fig. 5.3.g Registro STATUS ……66

Fig. 5.3.h Registro de funciones especiales del PIC16f886 …… . 67

Fig. 5.3.i Resumen del Banco 2 del Registro de funciones especiales del PIC16f886 …… 69

Fig. 5.3.ii Resumen del banco 3 del Registro de funciones especiales del PIC16f886 …… 69

Fig. 5.3.j Sentencia Switch …… 73

Fig. 5.3.k Pin de activación del latch a utilizar …… 74

CAPITULO 6: PRUEBAS Y CONCLUSIONES

Fig. 6.1.a Sistema de comunicación paciente- personal de enfermería …… 77

Fig. 6.1.b Prototipo del dispositivo del paciente …… 78

CAPITULO 1: INTRODUCCIÓN

1.1 FUNCIONALIDAD DE UN SISTEMA DE COMUNICACIÓN PACIENTE-PERSONAL DE ENFERMERÍA DENTRO DE UN HOSPITAL

La estancia dentro de un hospital, de un paciente nunca ha sido demasiado placentera, es por eso que en cuanta mejor atención y comodidad se brinde el paciente tendrá una estancia más tranquila y de mejor calidad, que por supuesto repercute en la salud misma.

El contacto humano del personal de enfermería con el enfermo es uno de los requisitos fundamentales para el bienestar del paciente, quien espera que se le dedique tiempo y atención exclusiva cuando lo necesita, la eficiente y correcta atención ayuda a disminuir el estrés del paciente.

Sin embargo estas demandas de atención en el servicio exigen más tiempo al personal de enfermería y por tanto incrementan la carga de trabajo diario, además, la pérdida de tiempo en vueltas innecesarias y tareas duplicadas incomodan tanto a las enfermeras como a los pacientes, haciendo que la calidad de atención disminuya.

He aquí que dentro de un hospital un buen sistema de comunicación entre el paciente y el personal de enfermería representa una herramienta útil para eliminar las vueltas innecesarias que repercute en los tiempos de espera del paciente al atender las llamadas que lo requieran con más prontitud.

Anteriormente estos sistemas electrónicos eran demasiados complejos porqué implicaban una red demasiado amplia de cableado para lograr la comunicación ya que cada instrucción ó llamada se hacía por una línea independiente, que a su vez, genera problemas al momento de surgir una falla en los dispositivos y querer repararla entre tantas líneas existentes en la red.

fig. 1.1.a. Módulo de pacientes del Hospital de "Venados" del IMSS

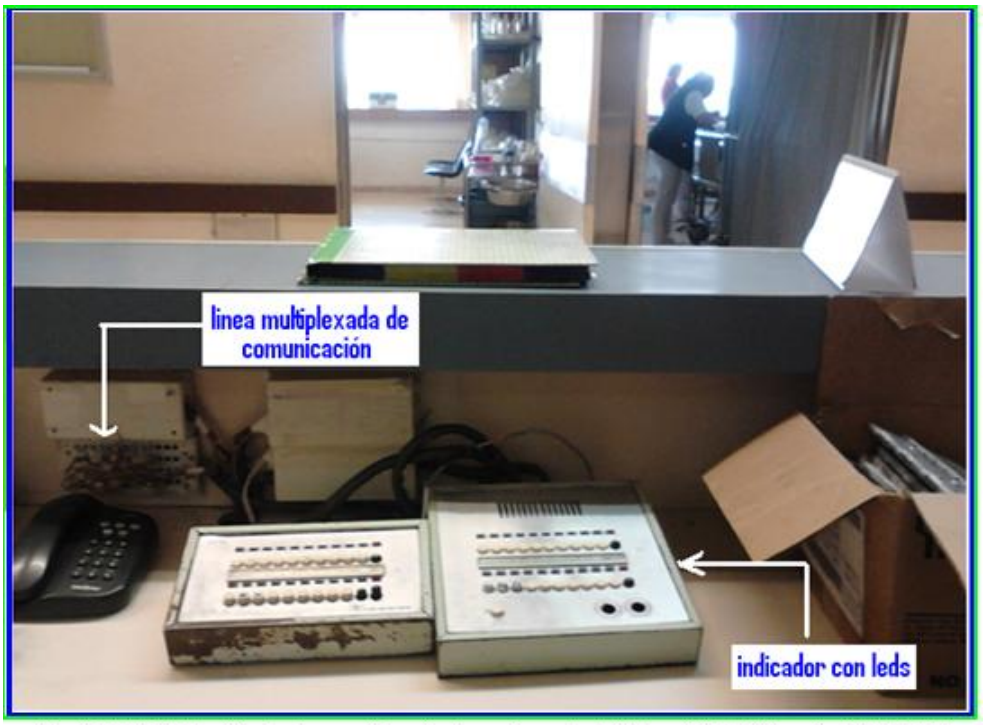


Fig. 1.1 b. Módulo eléctronico en el contro de enfermería del Hospital de "Venados" del IMSS

Por tal motivo, en esta tesis se propone implementar un sistema sencillo que elimine esa gran cantidad decableado, y que facilite un mantenimiento preventivo o correctivo del mismo, además de ser eficiente y económico. Pero sobre todo que le sea útil y de fácil manejo para el paciente.

El modo operativo de éste sistema de intercomunicación entre el paciente y enfermería comienza al iniciarse una llamada desde una habitación, encendiéndose el indicador led correspondiente al tipo de llamada, del modulo electrónico colocado sobre la pared del paciente.

En la estación central se activan las señales apropiadas. El modulo electrónico de la estación central incorpora una memoria para guardar todas las llamadas realizadas hasta ser atendidas; La anulación de la llamada se realiza desde la habitación mediante un pulsador, una vez se haya atendido al paciente.

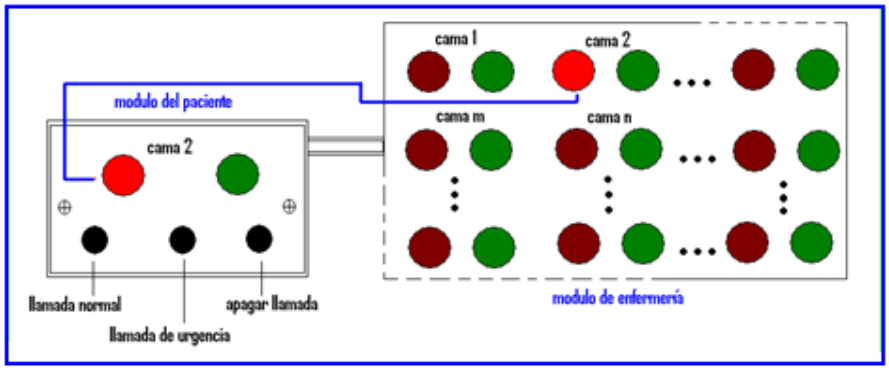


fig. 1.1 .c Diagrama de comunicación entre módulos

La función principal de este sistemaesmantener en constante comunicación,al paciente con elpersonal de enfermería permitiendo a este último asistir al paciente de acuerdo al tipo y tono de llamada que segeneradesdela habitación.

1.2. CUADRO COMPARATIVO DE BUSES SERIALES (RS232, SPI, CAN, LIN, RS485).

Para minimizar el número de conductores a utilizar, lo más conveniente es que la información que se vaya a transmitir no se haga en paralelo enviando cada bit por línea, sino en serie, es decir que independientemente de la instrucción que se dé, dicha información viaje por la misma línea enviando un bit detrás de otro.

Para lograr este objetivo nos apoyaremos de los sistemas electrónicos digitales, específicamente de la programación de microcontroladores, formando una red de microcontroladores, además de un bus serial y un protocolo serial para nuestro bus.

Este sistema propone que un conjunto de microcontroladores apunten exclusivamente al que se encuentra en el control de enfermería, no necesitan comunicarse entre ellos. Esta condición puede satisfacerse de dos maneras:

- La primera es asignando una dirección a cada microcontrolador por cubículo de paciente, convirtiéndolos en esclavos y enviando el dato de la petición del paciente a un solo maestro que será el que se encuentra en el control de enfermería (maestro/multiesclavos).

- La segunda manera es designando como maestros a todos los microcontroladores por cubículo de paciente y que todos ellos apunten a un solo esclavo que será el que está en el control de enfermería (multimaestro/esclavo).

Hay diversos tipos de protocolos de comunicación en serie que podemos usar, entre los que encontramos los siguientes:

- SPI (Bus Serial de Interfaz Periférica)
- I2C(Circuito Inter-integrado)
- RS- 232 (Estándar Recomendado 232)
- RS-485(Estándar Recomendado 485)
- LIN (Internet de Red Local)
- CAN (Control de Área de Red)

El siguiente cuadro comparativo fue diseñado con el objetivo de ver las características que ofrece cada uno de los métodos posibles a utilizar, si se desea profundizar en información concreta sobre un método en particular, se ponen en las notas al pie de página los vínculos de donde se obtuvo dicha información. De manera resumida se explicara cómo funciona cada método a fin de explicar porque se elige el I2C.

En general los dispositivos que transmiten en forma síncrona son más caros que los asíncronos debido a que son más sofisticados en el hardware para proveer de la señal de reloj que establecerá la velocidad de transmisión de datos. En contraparte los que trabajan en forma asíncrona trabajan con software a través de bits especiales dentro del código de programación para la sincronización. Sin embargo algunas otras características que ofrecen los dispositivos que transmiten en forma asíncrona los harán aun más caros que los primeros como veremos posteriormente.

- SPI (Bus Serial de Interfaz Periférica)

El bus SPI será el primero en ser descartado porque aun cuando nos puede ser útil en la configuración maestro/multiesclavo, no disminuye las líneas de comunicación ya que necesita una línea extra denominada Chipselect (SS) dependiendo de la cantidad de dispositivos con los que se desea tener comunicación.

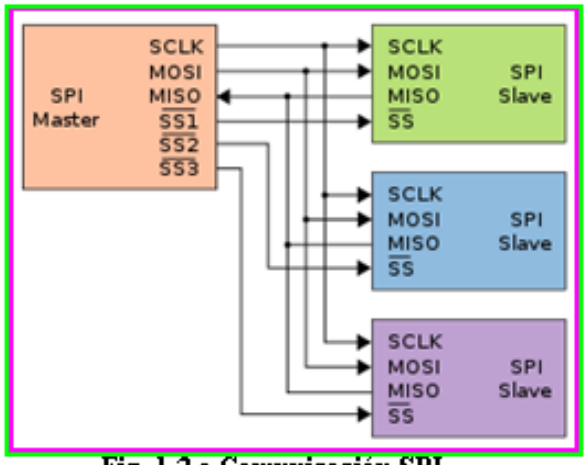


Fig. 1.2.a Comunicación SPI

- RS- 232 (Estándar Recomendado) y RS-485

El RS-232 solo utiliza tres líneas y establece una conexión punto a punto, es decir solo sirve para dos dispositivos. En contraparte el bus RS-485 soporta la configuración multimaestro, éste funciona mediante una diferencia de voltajes entre sus líneas de transmisión, sin embargo si quisiéramos implementar un bus con solo dos líneas solo podemos conectar al mismo bus 32 dispositivos, este número puede aumentar a 64 con el inconveniente de que el bus utilice 4 líneas.

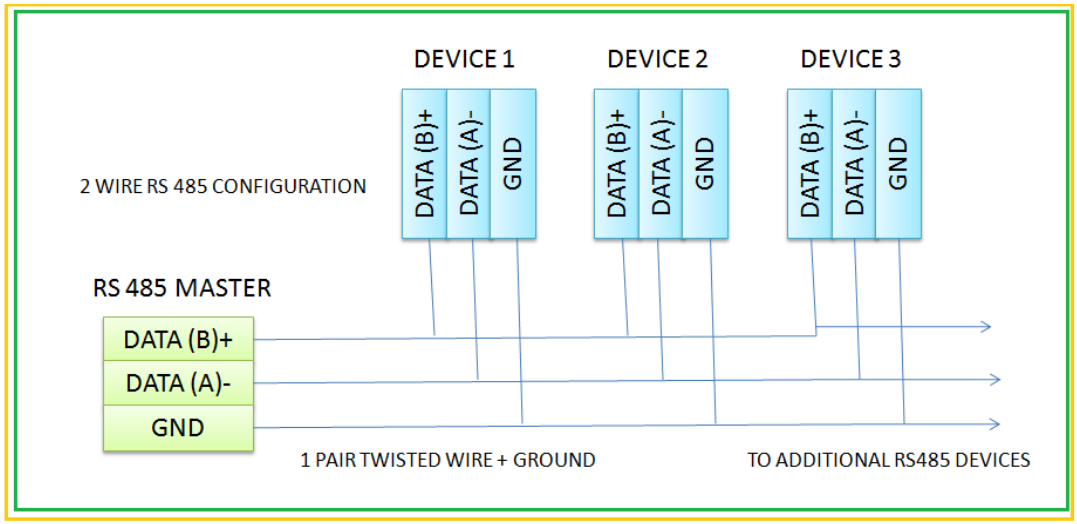


Fig. 1.2.b Comunicación rs-485

- LIN (Internet de Red Local)

El protocolo LIN aunque solo utiliza una línea para la transmisión de datos, presenta dos inconvenientes importantes, primero, no acepta la configuración multimaestro y segundo solo es útil para16 unidades esclavas.

- CAN (Control de Área de Red)

El protocolo CAN (control de área de red) parece ser de los más completos, éste trabaja solo con dos líneas: CANH y CANL, al igual que con el RS-485 basa su funcionalidad por la diferencia de tensiones entre líneas y por tanto es inmune al ruido; La distancia de transmisión es bastante en comparación con todos los demás buses de transmisión; a diferencia del I2C no utiliza direcciones sino mensajes, el nodo emisor transmite el mensaje a todos los nodos de la red sin especificar un destino y todos ellos escuchan el mensaje para luego filtrarlo según le interese o no, además,al igual que el I2C, el Bus Can integra la detección y señalización de errores. Existen microcontroladores que traen un controlador CAN integrado sin embargo el transceptor (transceiver) no está dentro del propio microcontrolador por tanto debe existir una conexión física a través de los pines C1RX (recepción CAN del micro), C1TX (transmisión CAN del micro) del microcontrolador, con los pines RXD y TXD del transceptor. La función del transceptor es la de adecuar la señal de entrada del bus a la del sistema. Esta entrada de señal del bus se efectúa a través de las patillas CANH y CANL del transceptor.

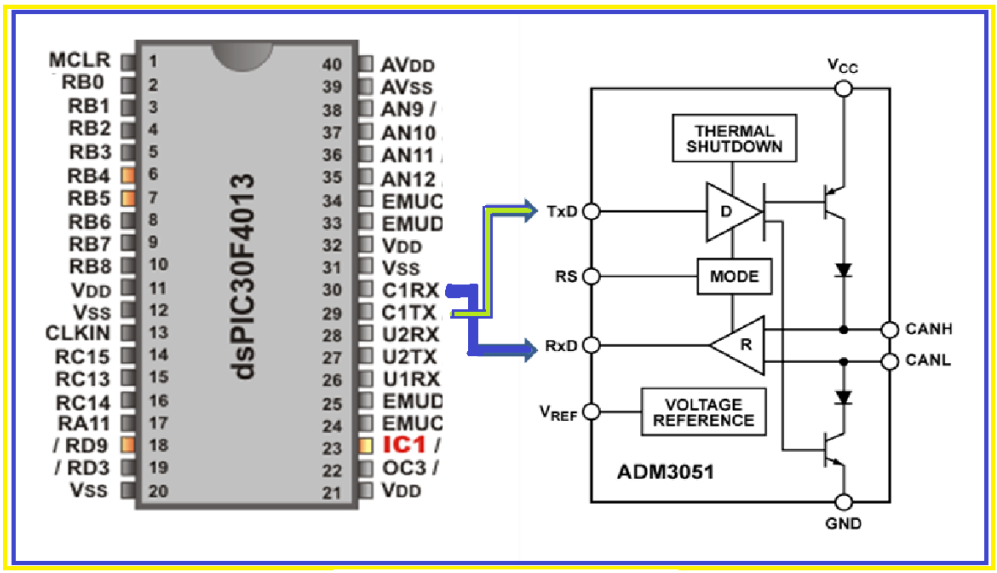


Fig. 1.2.c Comunicación CAN

Dadas las características de este tipo de comunicación, los microcontroladores que trabajan con este protocolo sobrepasan por mucho las características necesarias para este proyecto, a diferencia de un microcontrolador con las características suficientes para trabajar con I2C,lo que se ve reflejado en el costo.

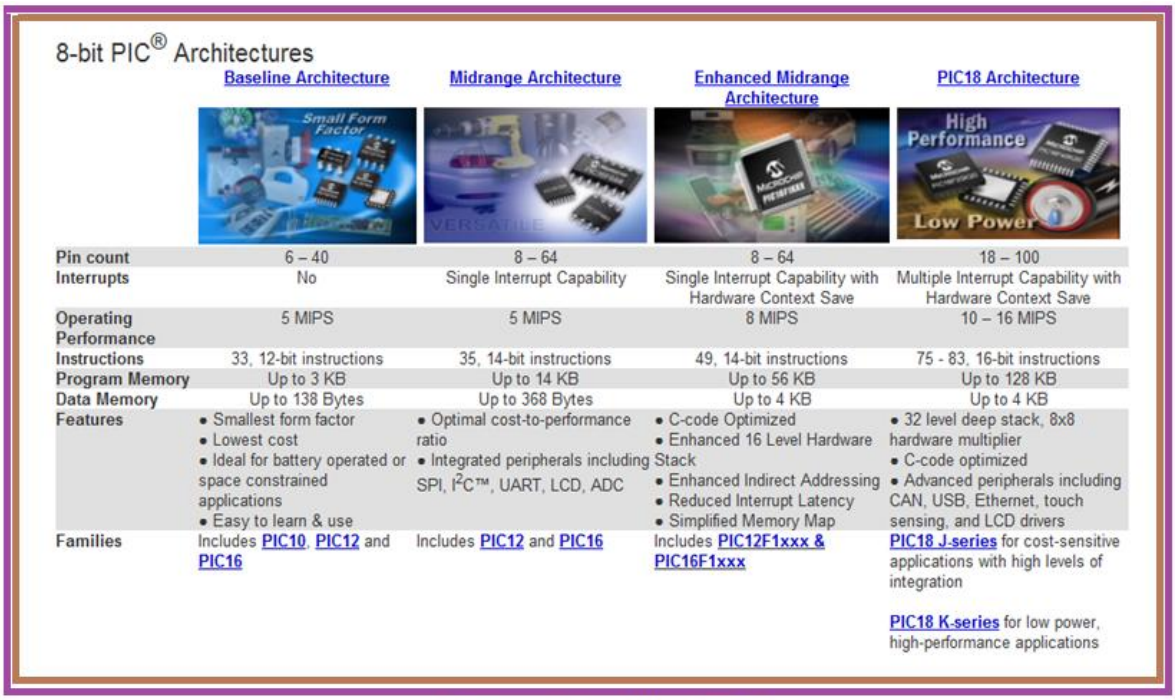


Fig. 1.2.d Familias de PICs

FIG. 1.2.e. TABLA DE COMPARACIÓN DE PROTOCOLOS

	SPI	I2C	RS232	RS485[1]	LIN	CAN
TIPO DE TRANSMISIÓN	Sincrono[2]*	Síncrono	Síncrona, asíncrona	Síncrona, asíncrona[3]	Asíncrono*	Asincrono[4]
SOPORTE MULTIMAESTRO	NO	SI	NO	SI	NO	SI[5]
PINES UTILIZADOS	SLCK, MOSI,MISO,SS	SDA,SCL	TxD,RxD,SG	TxD,RxD,SG	LIN BUS	CANH, CANL,
COMUNICACIÓN	Half dúplex, Full dúplex *	Halfdúplex, full dúplex*	simplex, half duplex o full dúplex[6]	HalfDuplex	Simplex*	Halfduplex
VELOCIDAD Y DISTANCIA DE TRANSMISION	320kbps-5Mbps[7],2 m	100kbps - 400kbps, y alta velocidad 3.4 Mbps	20 Kbps, hasta 15m[8]	10Mbps a 10 m, y 100kbps a 1200m[9]	20 kbps [10], 40 m y aumenta con un repetidor[11]	125 Kbps a 500m -1 Mbps a 40 m[12]
CONSUMO DE ENERGIA	No utiliza Pull-ups y ahorra energía, fuente +5V	Utiliza Pull-Ups, fuente +5V	+12 y -12 Volts	fuente +5V , 1.5v entre las salidas diferenciales.	Utiliza Pull-Ups	cable L (Low) = 0V y 2.25V, cable H= 2.75V. y 5V
DIRECCIONAMIENTO	utiliza líneas especificas para cada dispositivo	Cada dispositivo tiene una dirección única por software, 128	Es punto a punto		basado en mensajes, no en direcciones	basado en mensajes, no en direcciones

		direcciones				
CANTIDAD DE DISPOSITIVOS SOBRE EL MISMO BUS	Depende de los pines disponibles del PIC para SS	Depende de la capacitancia de las líneas 400 pf	Uno a uno	32 (Bus de 2 hilos), 64 (4 hilos bus)	16 unidades esclavas[13]	127
NOTAS			Trabaja con lógica negativa			Necesita de un transeiver x cada dispositivo
COMPAÑIA	Motorola	Philips Semiconductor	Electronic Industries Alliance	Electronic Industries Alliance		Robert Bosch GmbH

1.3 ¿POR QUÉ I2C?

Finalmente llegamos al protocolo I2C, éste solo utiliza dos líneas, una línea es la encargada de transmitir la señal de reloj llamada SCL(serial clock) y la línea llamada SDA(serial data) que proporciona los datos de transferencia. Éste protocolo determina la cantidad de dispositivos interconectadospor las direcciones que se pueden conformar con 7 bits, lo cual nos da la posibilidad de direccionar hasta 127 dispositivos, y no por la cantidad de pines de un PIC disponibles para cada dispositivo. Sin embargo el número máximo de nodos no solo está limitado por el espacio de direcciones sino también por la capacitancia total de los buses con un límite de capacitancia entre líneas de 400pF, lo que restringe la comunicación a distancias de unos cuantos metros.

Aun cuando la mínima velocidad de transmisión (100kbps) es suficiente para este proyecto, I2C fue diseñado para distancias cortas, como por ejemplo dentro de lamotherboard de una computadora, sin embargo el problema de la distancia puede ser solucionado utilizando un circuito MAX232 que adapta la señal para la transmisión a 100 metros. Además I2C acepta el modo de operación mutiesclavos- maestro y multimaestro-esclavo, siendo este último el modo de operación de nuestra red. A esto último le agregamos que si un dispositivo de la red deja de funcionar, sea cual sea la causa, se le puede omitir, o de igual forma se le pueden añadir módulos al bus sin afectar al resto del sistema y sin necesidad de reprogramación. Por dichas características, éste bus es una buena opción para nuestro proyecto.

1. http://www.pantallas-electronicas.es/index.php/es/maes-lejos-con-rs485.html
2. http://www.ehowenespanol.com/protocolos-sincronos-asincronos-lista_43724/
3. http://www.piclist.com/techref/io/serials.htm
4. http://www.iiis.org/CDs2009/CD2009CSC/CISCI2009/PapersPdf/C989WT.pdf
5. http://www.ucpros.com/work%20samples/Microcontroller%20Communication%20Interfaces%203.htm
6. http://es.wikipedia.org/wiki/RS-232
7. http://www2.ate.uniovi.es/fernando/Doc2007/Presentaciones/Serie_sincrona_en_C.pdf
8. http://www.cimco.com/docs/cimco_dnc-max/v6/es/#SerialComStandards
9. http://www.epanorama.net/links/tele_interface.html
10. https://sites.google.com/site/sistemasdemultiplexado/protocolos-de-comunicacin/lin-bus
11. http://www.ucpros.com/work%20samples/Microcontroller%20Communication%20Interfaces%203.htm
12. http://uajfk2002.tripod.com/gj1/red-can.htm
13. http://www.dacarsa.net/formacion.php?p=1&b=77

CAPITULO 2: BUS I2C

2.1 EL BUS I2C

La siglaI2Cproviene de "Inter IntegratedCircuit" y es un tipo de interfaz serial estándar que utiliza 2 cables, esta interfaz fue diseñada por Philips Semiconductors al comienzo de los años 80's. El objetivo de este proyecto era comunicar un CPU y los periféricos de unaTVcon el mínimo de cableado. El BUS físicamente consiste en 4 líneas, unalínea para la alimentación, otra para la conexión a tierra y dos líneas más que se describen a continuación:

- SDA (Serial Data line): es la línea para la transferencia serie de los datos
- SCL (Serial Clock Line): es la línea para transmitir la señal de reloj que genera el dispositivo maestro y que se utiliza para la sincronización de los datos.

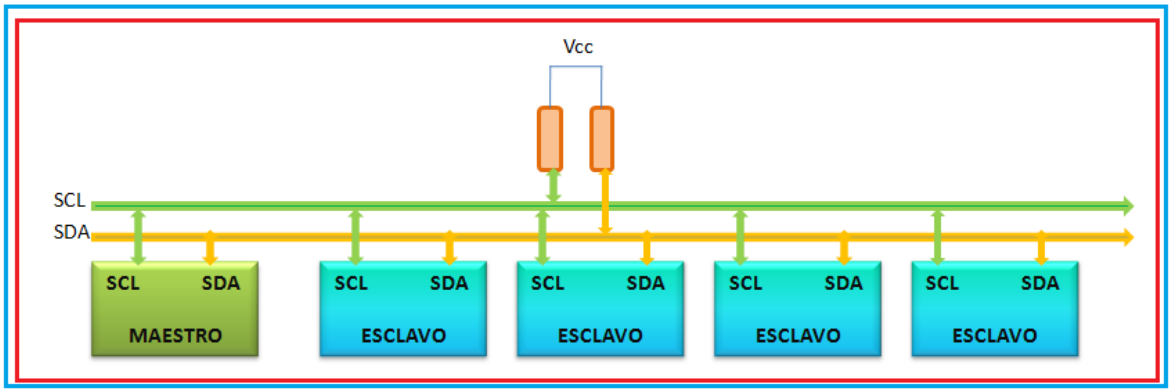


Fig. 2.1.a Estructura del BUS I2C

Ambas líneas resultan ser bidireccionales y trabajan solo con dos estados; activo - bajo ó pasivo – alto, además de que pueden realizar una comunicación full duplex entre los dispositivos conectados al circuito. Los datos pueden ser transferidos a una velocidad que supera los 100kbps en el modo estándar y sobre los 400kbps en el modo Fast, y sobre 3.4Mbps en el modo High Speed. En el modo Standard se utilizan 7-bit para la direccionar los dispositivos en los otros modos de transferencia los dispositivos esclavos pueden utilizar 7 ó 10bits.

Cada dispositivo perteneciente al bus tiene una dirección de memoria única y puede trabajar como receptor y/o transmisor dependiendo de cómo haya sido configurado; los dispositivos compatibles con I2C suelen llevar 2 o 3 pines para poder modificar esta dirección ó bien en el caso de un microcontrolador puede asignársele una dirección al mismo mediante programación, esto a modo de que el diseñador evité que en un mismo diseño haya dos ó más esclavos/maestros con la misma dirección.

El protocolo del bus I2C soporta detección de colisiones*, sincronización de reloj y hand-shaking* (normas de comunicación)para múltiples dispositivos conectados en el

bus.Además soporta más de un maestro conectado al mismo bus. Los pulsos del reloj son siempre generados por la unidad maestra.

El máximo número de unidades que pueden conectarse en un bus I2C está definido por la máxima capacitancia de la línea de 400pF, y el límite para direccionara los dispositivos; la capacitancia típica de un dispositivo es de 10pFy el número de combinaciones posibles para direccionar a un dispositivo esclavo es de 2^7 = 128 combinaciones.

En resumen la comunicación se realiza de la siguiente manera: el dispositivo maestro pone en el bus la dirección del dispositivo con el que se quiere comunicar. Todos los dispositivos monitorean el bus para determinar si el maestro está enviando su dirección. De esta forma solo el dispositivo con la dirección correcta puede comunicarse con el maestro.

2.2 HARDWARE DEL BUS I2C

El Hardware del bus I2C se basa en la lógica de la compuerta AND. La configuración de salida de los dispositivos conectados al bus deben ser drenador abierto ó colector abierto para poder realizar precisamente la función deAND cableada.

Las características de este tipo de configuración son:

- La salida de la compuerta es directamente uno de los "colectores" o "drenadores" con que está construida la compuerta, por eso se dice que es una configuración de colector o drenador abierto.
- Es necesaria una resistencia de "carga" o "pull up" a la salida para poder tener una respuesta de la compuerta.
- Permite conectar varias fuentes de datos a un mismo hilo
- Se dice que hay un nivel alto en el bus si:
  - Ningún dispositivo accede al bus
  - Si ningún dispositivo transmite un cero
- Se dice que hay un nivel bajo en el bus si:
  - Si un dispositivo pone un nivel bajo
  - Si dos dispositivos escriben a la vez siempre prevalecen los ceros

Si las líneas SDA y SCL de entrada están conectadas a tensión positiva a través de resistencias pull-up, la salida de la compuerta será un uno lógico y de esta manera se forma una lógica AND cableada, mientras que cualquier otro caso arrojara un cero a la salida.

Recordando la tabla de verdad de la lógica AND decimos que solo se produce un nivel alto si todas las entradas son altas, de lo contrario la salida estará en "bajo". De ésta forma si se conectan entre si todas las salidas de las compuertas de colector abierto, a través de las resistencia pull-up a la alimentación, la salida común solo será "alta" cuando todos los transistores de salida estén "apagados" o en "corte"; en éste caso se dice que el bus está libre.Si un solo transistor de salida "conduce" o entra en estado de "saturación", hace que la

salida pase al estado bajoindependientemente del estado de los otros transistores y el bus queda ocupado a nivel bajo.

El cálculo de las resistencias de Pull-up depende de la tensión de alimentación, de la capacidad del bus y del número de dispositivos conectados. Esto se tabula en unas tablas que facilita el fabricante Philips Semiconductors. Un valor de 4.7 Kohms es satisfactorio para la mayoría de las aplicaciones.

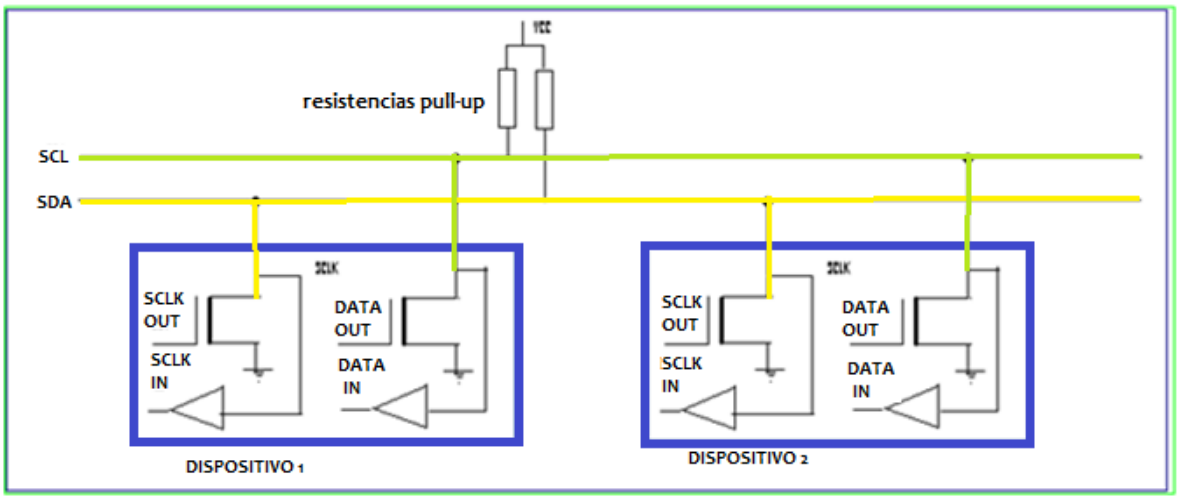


Fig. 2.2.a Etapa de salida de los dispositivos I2C

El inconveniente de la conexión en colector abierto es que a través de las resistencias pull-up se genera una capacitancia entre líneas que deforma la señal SDA es por eso que el número de dispositivos conectados al bus está determinado por la capacitancia entre líneas a 400pF, sin embargo esto puede ser solucionado utilizando una carga activa* en lugar del resistor.

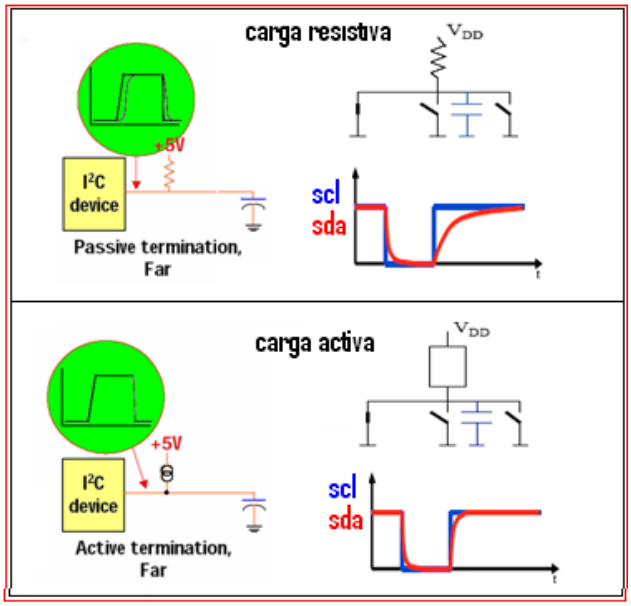


fig. 2.2.b Capacitancia entre lineas

2.3 TRANSFERENCIA DE UN BIT POR LA LÍNEA SDA

Para transferir un bit por la línea de datos SDA se debe generar un pulso de reloj por la línea SCL. Los bits de datos transferidos por la línea SDA deben mantenerse estables mientras la línea SCL esté a nivel alto. El estado de la línea SDA sólo puede cambiar cuando la línea SCL está a nivel bajo.

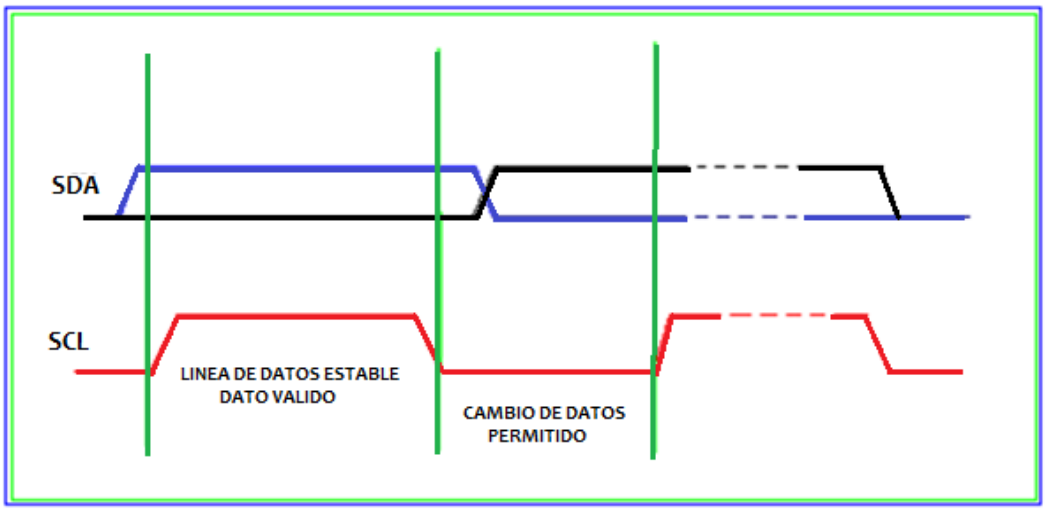


Fig. 2.3.a Transferencia por la linea SDA

Si la línea SDA cambia mientras SCL está a nivel alto, no se interpreta como dato, sino como una condición especial (Start o Stop).

2.4 CONDICIONES DE START Y STOP

Para que la transferencia de información pueda ser iniciada el bus no debe estar ocupado. Esto quiere decir que los transistores de salida de todos los dispositivos conectados al bus I2C deben de estar en alta impedancia. Para indicar que el bus está libre las líneas del reloj SCL y datos SDA deben estar a nivel alto. Una vez que se ha verificado esto, el transmisor procederá a enviar un bit cada pulso del reloj.

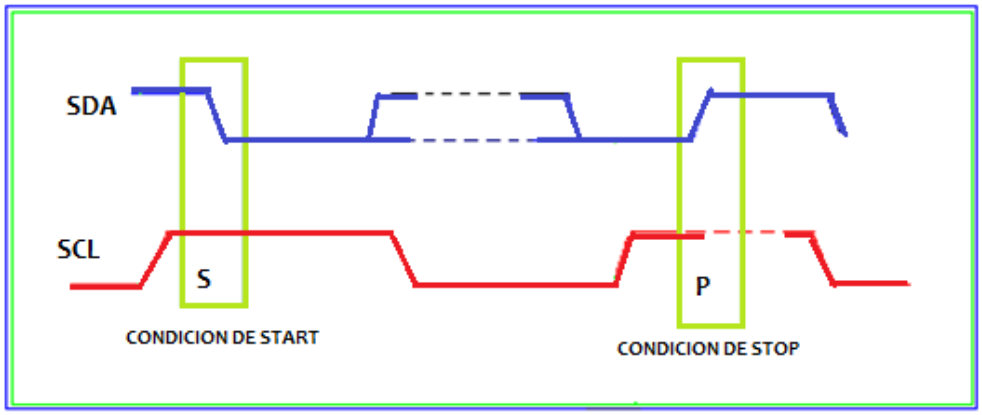


Fig 2.4.a Condiciones de Start y Stop

En la transmisión de datos se da la siguiente secuencia de sucesos:

1.- Condición de start. SDA debe estar en flanco de bajada mientras que SCL permanece a nivel alto. Esta condición señala el comienzo de la transferencia de datos.

2.- Transmisión de los bits de información a cada pulso del reloj.

3.- Condición de stop. SDA en flanco de subida mientras que SCL se encuentra a nivel alto. Ésta es la condición que indica el fin de la transferencia.

Las condiciones de start y stop son siempre generadas por el microcontrolador maestro. El bus I2C se considera ocupado después de la condición de start. El bus se considera libre de nuevo después de un cierto tiempo tras la condición de Stop. Los dispositivos esclavos compatibles con bus I2C deben poseer el hardware de acoplamiento necesario que le permita detectar las condiciones de start y Stop.

2.5 TRANSFERENCIA DE DATOS

Cada dato enviado por la línea SDA debe tener 8 bits (1 byte). El número de bytes que se pueden enviar no tienen restricción. El byte de datos se transfiere empezando por el bit 7 que es de mayor peso, denominado MSB (Mosts ingnificant bit).

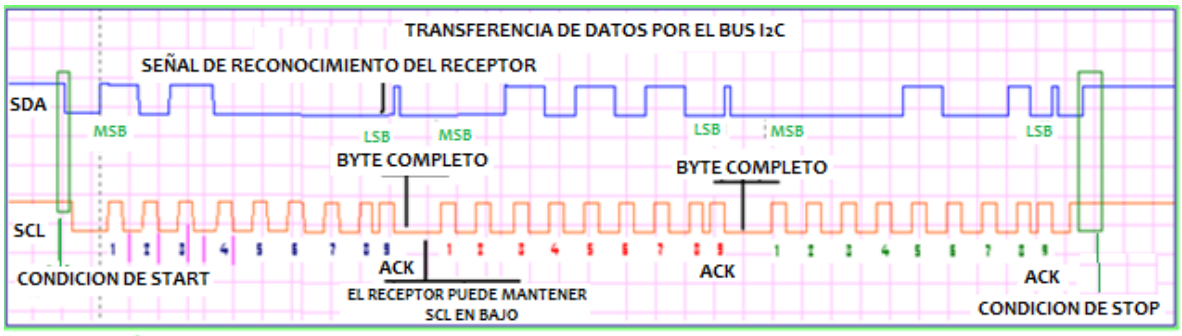


Fig. 2.5.a Transferencia de datos

Una vez que se han transmitido los 8 bits el receptor deberá mandar un bit de asentamiento o reconocimiento (acknowledgement) en el noveno pulso del reloj. El receptor ejecuta este reconocimiento poniendo la señal SDA a un nivel bajo. Cada grupo de 8 bits debe ser asentido.

El bit de reconocimiento ACK es obligatorio en la transferencia de datos. Si el esclavo-receptor que esta direccionando no desea recibir más bytes el maestro debe detectar la situación y no enviarle más datos. Esto se indica porque el esclavo no genera el bit ACK del último byte quedando la línea SDA a nivel alto, lo cual es detectado por el maestro que puede generar la condición stop y aborta la transferencia de datos.

Lo que sucede en el noveno pulso del reloj con el bit de reconocimiento es lo siguiente (Fig. 2.5.b):

El receptor que va a dar un acuse de recibido ACK pone a nivel bajo la línea SDA inmediatamente después de la recepción del octavo bit. Esto significa que tan pronto como

el transmisor baja SCL para completar la transmisión del bit (1), SDA será puesto a bajo por el receptor (2).

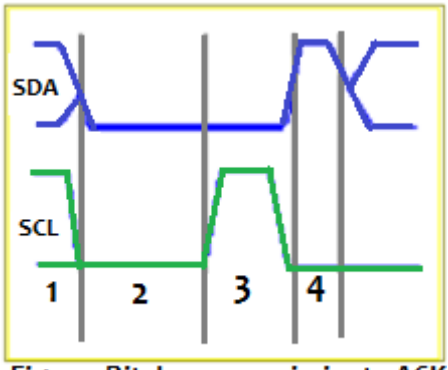


Fig 2.5. Bit de reconocimiento ACK

El maestro-transmisor ahora emite un pulso de reloj en la línea SCL (3) El esclavo- receptor liberará la línea SDA hasta la terminación de este pulso (4) de reloj. De esta forma el Bus queda disponible de nuevo para que el maestro continúe enviando datos o generar una condición de STOP.

Sin embargo no es imprescindible enviar una condición de stop para abortar una transferencia de datos, si se repite la condición de Start se aborta la transferencia de datos anterior y se comienza una nueva.

2.6 FORMATO DE TRANSFERENCIA DE DATOS

Los datos transferidos tienen la siguiente forma, la siguiente figura es la señal simulada en PROTEUS. En la misma se aprecia que para operar un esclavo sobre el bus I2C son necesarios seis pasos para enviar o recibir información:

1. Un bit de Start, que señaliza el inicio de la transferencia de datos.
2. Siete bits de direccionamiento de un esclavo
3. Un bit de lectura/escritura (R/W) que define si el esclavo es transmisor o receptor.
4. Un bit de reconocimiento ACK
5. Un mensaje dividió en bytes(8bits)
6. Un bit de stop que indica el fin de la comunicación

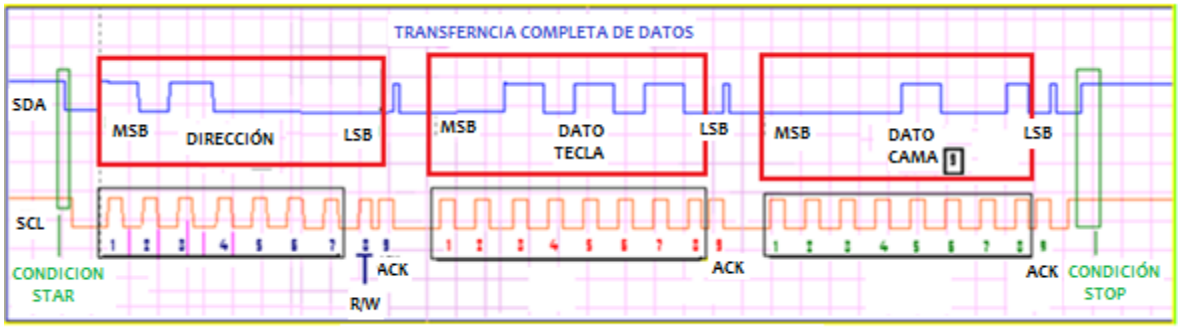


FIg. 2.6.a Transferencia completa de datos

Debido a que puede haber varios esclavos conectados al bus, el maestro debe indicar mediante una dirección de esclavo a cuál de ellos va destinada la información a transferir, (slave address). Cada esclavo tiene una única dirección con la que es identificado. Todos los esclavos están escuchando continuamente la línea para reconocer si es a ellos a quien se dirige el maestro. El procedimiento de dirección para el bus I2Cestablece que el primer byte después de la condición de start determina el esclavo

seleccionado por el maestro. En nuestro caso tenemos varios maestros y todos apuntando a una misma dirección del esclavo.

Los primeros siete bits indican la dirección del esclavo el octavo bit (R/W) determina en qué dirección viajan los datos:

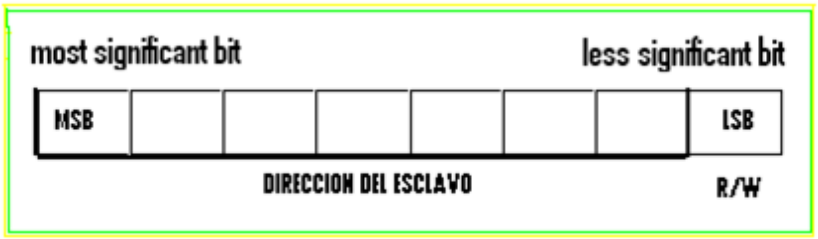


Fig 2.6.b Direccionamiento del esclavo

Si R/W=0, el esclavo es receptor. Significa que el maestro escribirá información en el esclavo seleccionado

Si R/W=1, el esclavo es emisor. Significa que el maestro leerá información del esclavo seleccionado

Cuando un microcontrolador maestro envía una dirección después de la condición de start cada dispositivo comprueba los siete primeros bits de la dirección con la suya propia. El que coincida se considera el dispositivo seleccionado por el maestro, que será un esclavo receptor o un esclavo emisor dependiendo del bit R/W. En la figura 2.6.a se observa que el esclavo es solamente receptor puesto que el bit R/W=0.

2.7 TIPOS DE FORMATOS DE TRANSFERENCIA

Los posibles formatos de transferencia son:

1.- Maestro-emisor que transmite al esclavo-receptor. Si el bit 8 es de escritura, (R/W=0), la secuencia será la siguiente:

El microcontrolador maestro inicia la condición de start seguida de la dirección del esclavo con el que se desea la comunicación (7 bits), si el octavo bit es 0 se escribirá un dato en el esclavo, después de cada 8 bits el microcontrolador maestro debe esperar una señal de reconocimiento ACK como respuesta por parte del esclavo en el noveno bit; Durante este proceso el maestro- transmisor lee el estado de la línea SDA, si el esclavo-receptor deja la línea SDA en nivel bajo el estado de transferencia continua hasta que el esclavo ya no desee recibir bytes de datos y cambie a nivel alto la línea SDA y de esta forma el maestro genere la condición de stop y se libere el bus I2C, las líneas SDA y SCL pasan a estado alto.

Fig. 2.7.a Transferencia maestro - emisor al esclavo - receptor

2.- Maestro-receptor lee de un esclavo-emisor inmediatamente después del primer byte. Si el bit 8 es un 1 lógico (R/W=1), la secuencia será la siguiente:

El microcontrolador maestro inicia con la condición de start, seguida de la dirección del esclavo con el que se desea la comunicación (7 bits), si el octavo bit es 1, por tanto el maestro recibirá un dato del esclavo, después de cada ocho bits el maestro debe generar en el noveno bit un bit de reconocimiento (ACK) hasta que se quiera finalizar la transferencia de datos y de esta forma el maestro deje de generar el ACK, así el esclavo–transmisor debe permitir desbloquear la línea SDA poniéndola a nivel alto para que el maestro genere la condición de stop.

Fig. 2.7.b Transferencia maestro - receptr al esclavo - emisor

2.8 CONEXIÓN DEL BUS I2CENTRE PIC 16F628A Y PIC16F886

La siguiente imagen muestra la conexión de tres microcontroladores 16f628A configurados como maestros, que utilizan las líneas RB0 y RB1 como líneas del bus I2C, y un microcontrolador configurado como esclavo (PIC16f886) que utiliza las líneas RC3 y RC4 como SCL y SDA respectivamente. Para que funcione correctamente el sistema es necesario que ambos microcontroladores acepten el protocolo I2C, esto se puede observar en las hojas de especificaciones.

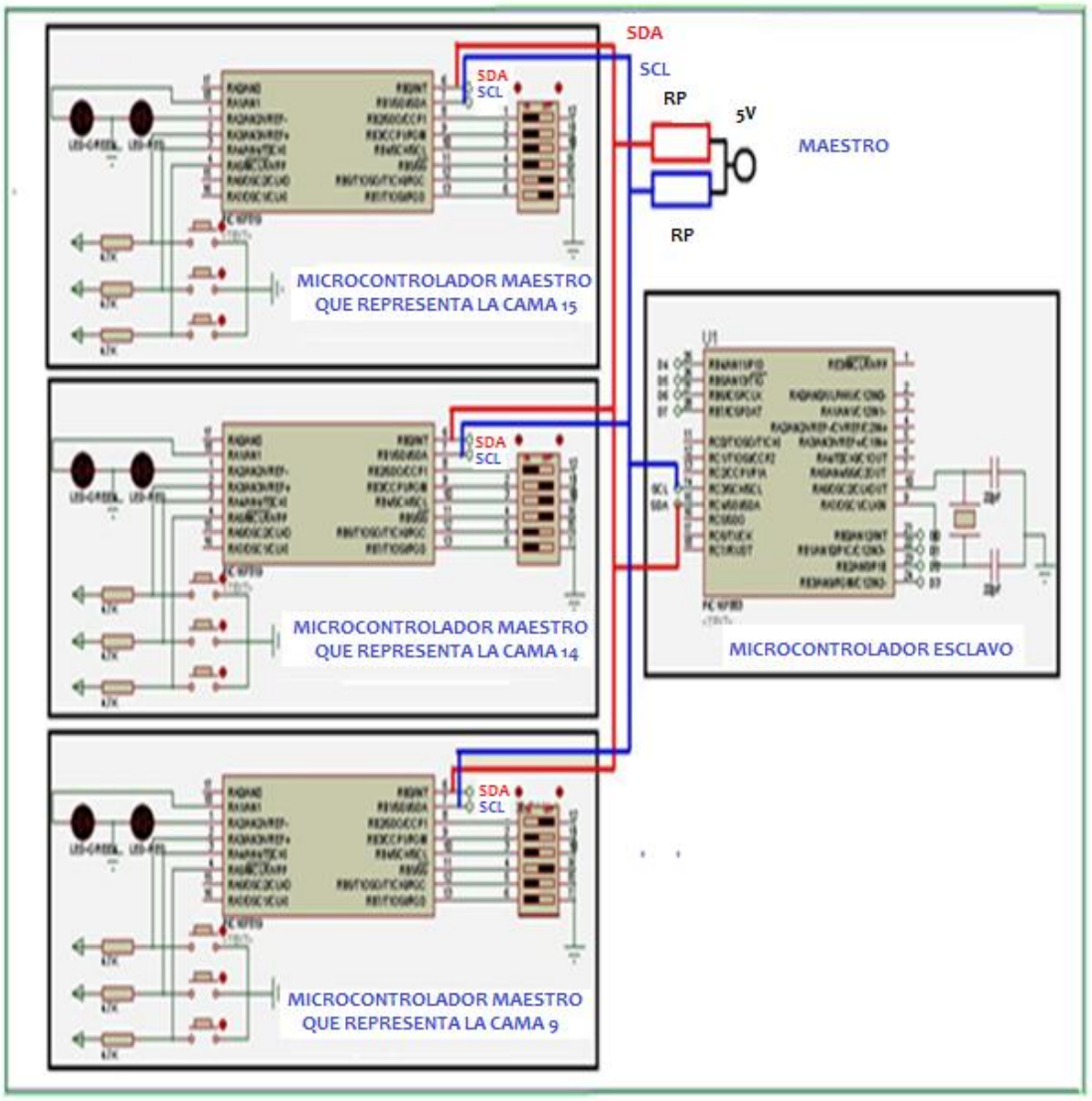


Fig. 2.8.a Diagrama de conexiones entre dispositivos maestros - emisor al esclavo - receptor

CAPITULO 3: PROGRAMACION DEL PIC

3.1 FUNCIONES BÁSICAS DE PIC-C PARA LA GESTIÓN DEL PROTOCOLO I2C

Según el software que utilicemos para programar el PIC contaremos con una librería de subrutinas para el bus I2C. Por en MPLAB, existen la librería BUS_I2C.INC que implementa la subrutinas de: “I2C_EnviaStart”, “I2C_EnviaStop”, “I2C_Enviabyte”, “I2C_LeeByte”.

Para esta tesis utilizamos PIC-C para programar nuestros PICs, al igual que MPLAB,PIC-C proporciona una librería llamada #use I2C las cual tiene efecto sobre las subrutinas, homologas a las de MPLAB, “I2C_START()”, “I2C_WRITE()”, “I2C_STOP()” , “I2C_READ()”.

Para llamar a esta librería en el programa se escribe la siguiente instrucción: #use I2C (opciones), y dentro de los paréntesis podemos definir una serie de opciones, que según el manual de ayuda del software es el siguiente:

Nota: las opciones deben ir separadas por comas

MASTER	Configura al PIC como maestro
MULTI_MASTER	Configura al PIC como multimaestro
SLAVE	Configura al PICcomo esclavo
SCL=pin	Especifica el pin utilizado para la línea SCL
SDA=pin	Especifica el pin utilizado para la línea SDA
ADDRESS=nn	Especifica la dirección del dispositivo esclavo
FAST	Especifica el modo rápido del protocolo I2C
FAST=nnnnnn	Configura la velocidad a determinada frecuencia en hertz
SLOW	Especifica el modo lento del protocolo I2C
RESTART_WDT	Reinicia el WDT mientras se espera I2C_READ
FORCE_HW	Usar hardware para las funciones I2C.
FORCE_SW	Usar software para las funciones I2C.Estas se usan cuando el PIC no dispone de hardware como el modulo MSSP (master synchronous serial port) y se intenta emular.
STREAM=id	Un Stream es un apuntador o identificador de flujo, el identificador puede ser utilizado en funciones como I2C_start(),I2C_read o I2C_write. Por ejemplo, si la función I2C_start() puede enviar una señal I2CstartóI2Crestart, se puede elegir qué tipo de señal necesitamos con configurar el valor del stream: un 2 para forzar un reinicio. Un valor de 1 hace un inicio normal. Si el reinicio no es especificado ó es 0, entonces un reinicio se hará solo si el ultimo compilador

encuentra un I2C_start() y no I2C_stop().

Otra aplicación resulta en un mismo PIC donde se desean dos buses I2C, mediante el stream se pueden dar a los puertos I2C diferentes identificadores de flujo y así es más fácil de diferenciar en el código qué puerto se está utilizando.

Ejemplos:

```
/*Configuración de dos puertos I2C, cada uno con un nombre de stream diferente*/

#use I2C(sda=PIN_C4, scl=PIN_C3, stream=I2C_HW)
#use I2C(sda=PIN_B1, scl=PIN_B2, stream=I2C_SW)

/*
La siguiente función leeun byte de datos del la memoria 24LC16 I2CEEPROM, usando el streamI2C_HW
*/

int Read2416(int address)
{
  I2C_start(I2C_HW, 1); //ejecuta un inicio
  I2C_write(I2C_HW, 0xA0);
  I2C_write(I2C_HW, address);
  I2C_start(I2C_HW, 2); //ejecuta un reinicio
  I2C_write(I2C_HW, 0xA1);
  data=I2C_read(IC2_HW, 0);
  I2C_stop(I2C_HW);
  return(data);
}
```

La manera en que se utiliza la librería #USE I2C es la siguiente:

1.-#use I2C(SLAVE, SDA=PIN_C4, SCL=PIN_C3, SLOW, ADDRESS=0xa0, NOFORCE_SW).

2.- #use I2C(MASTER, SDA=PIN_b0, SCL=PIN_b1, SLOW, NOFORCE_SW)

El ejemplo 1nos dice que vamos a configurar al PIC como esclavo, el pin 4 del puerto C se utilizara para la línea SDA y el pin 3 del puerto Cse utilizara para la línea SCL, la dirección del esclavo para este caso es 0x0a pero podemos asignarle cualquier otra recordando que tenemos disponibles 128 direcciones, a continuación se especifica el modo lento de operación del protocolo; si nos fijamos en la hoja de especificaciones del PIC16f819 vemos que para el bus I2C el modo rápido trabaja a 400 kHz y el modo estándar a 100 kHz.

Por último indicamos al PIC que el protocolo I2C no será implementado por software sino por hardware esto solo es posible si el micro controlador cuenta con un módulo SSP (synchronous serial port) ó un modulo MSSP (master synchronous serial port) para lo cual consultamos las hojas de características del microcontrolador a utilizar.

El ejemplo 2 configura al PIC como maestro donde el pin 0 del puerto B se utilizara para la línea SDA y el pin 1 del puerto B se utilizara para la línea SCL, se determina un modo de operación del protocolo lento y de igual forma el protocolo será implementado por hardware y no por software.

3.1.1 I2C_START();

Sintaxis:	I2C_start() I2C_start(stream) I2C_start(stream, restart)
Parámetros:	stream: especifica el identificador de flujo definido en #USE I2C restart: Si el restart es 2, un nuevo reinicio es forzado en lugar de un inicio. Si el restart es 1, se ejecuta un inicio normal. Si no se coloca ningún restart, el reinicio se hace solo si el último compilador encontró un I2C_START y no un I2C_STOP.
Retorno de la función:	indefinido
Función:	Emite una condición de inicio cuando el microcontrolador se encuentra en modo maestro I2C. Después de la condicióndeiniciola señal de reloj se mantiene en un valor bajohastaque se llama a la subrutina I2C_write (). Si otrasubrutinaI2C_start se llama en la mismafunción antes de llamar a unafunciónI2C_stop, entonces se emiteunacondición de reinicio especial.
Disponibilidad:	Todos los dispositivos.
Requerimientos:	#use I2C

Ejemplos:	I2C_start();
	I2C_write(0xa0); // dirección del dispositivo
	I2C_write(address); // dato enviado al dispositivo
	I2C_start(); // Reinicio
	I2C_write(0xa1); // cambiar la dirección del dato
	data=I2C_read(0); // ahora lee del esclavo
	I2C_stop();

3.1.2 I2C_WRITE();

Sintaxis:	I2C_write (***data***) I2C_write (stream, ***data***)
Parámetros:	***data*** - es una variable int de 8 bits **stream** - especifica el identificador de flujo definido en #USE I2C
Retorno de la función:	Esta función regresa el bit ACK. 0 significa ACK, 1 significa NO ACK, 2 significa que hubo una colisión si esta en el modo multi-maestro.
Función:	Envía un byte sobre la interfaz I2C. En el modo maestro ésta función genera una señal de reloj con el dato y en el modo esclavo espera por una señal del reloj del maestro. Esta función regresa un bit ACK. El LSB (bit menos significativo) del primer dato escrito después de la condición de inicio determina la dirección del dato transferido (0 es de maestro a esclavo y 1 de esclavo a maestro).
Disponibilidad:	Todos los dispositivos.
Requerimientos:	#use I2C

Ejemplos:	Longcmd;
	...
	I2C_start(); // condición de inicio
	I2C_write(0xa0);// dirección del dispositivo
	I2C_write(cmd);// byte bajo de comando
	I2C_write(cmd>>8);// byte alto de comando
	I2C_stop(); // Condición de paro

3.1.3 I2C_STOP();

Sintaxis:	I2C_stop() I2C_stop(stream)
Parámetros:	stream: especifica el identificador de flujo definido en #USE I2C
Retorno de la función:	indefinido
Función:	Emite una condición de paro cuando se encuentra en el modo maestro I2C.
Disponibilidad:	Todos los dispositivos.

Requerimientos:	#use I2C

Ejemplos:	I2C_start(); // condición de inicio
	I2C_write(0xa0); // dirección del dispositivo
	I2C_write(5); // comando del dispositivo
	I2C_write(12); // dato del dispositivo
	I2C_stop(); // condición de paro

3.1.4I2C_READ();

Sintaxis:	data = I2C_read();
	data = I2C_read(ack);
	data = I2C_read(stream, ack);

Parámetros:	***ack*** -Opcional, por default a 1. 0 indica un bit de noack. 1 indica un bitack. **stream** – específica el identificador de flujo definido en #USE I2C
Retorno de la función:	Dato tipo entero de 8 bits
Función:	Lee un byte sobre la interface I2C. En modo maestro ésta función genera una señal de reloj y en el modo esclavo espera por la señal de reloj del maestro. No existe un timeout* para el esclavo, se usa I2C_POLL para evitar un bloqueo. Se usa un RESTART_WDT en la directiva #USE I2C para forzar al contador watch-dog en el modo esclavo mientras espera.
Disponibilidad:	Dispositivos con interfaz I2C
Requerimientos:	#use I2C

Ejemplos:	I2C_start();
	I2C_write(0xa1);
	data1 = I2C_read();
	data2 = I2C_read();
	I2C_stop();

3.1.5I2C_POLL();

Sintaxis:	I2C_poll() I2C_poll(stream)
Parámetros:	stream (opcional)- específica el identificador de flujodefinido en #USE I2C
Retorno de la	1 (TRUE) ó 0 (FALSE)

función:	
Función:	La función I2C_POLL() solo debe ser usada cuando se use el módulo SSP. Esta función regresa TRUE si el hardware ha recibido un byte en el buffer. Cuando un TRUE ha sido devuelto, la llamada a la subrutina I2C_READ() inmediatamente regresara el byte que fue recibido.
Disponibilidad:	Dispositivos con módulo I2C
Requerimientos:	#use I2C

Ejemplos:	I2C_start(); // condición de inicio
	I2C_write(0xc1); // dirección del dispositivo/lectura
	count=0;
	while(count!=4) {
	while(!I2C_poll()) ;
	buffer[count++]= I2C_read(); //lee el siguiente dato
	}
	I2C_stop(); // condición de paro

3.2 DIAGRAMAS DE FLUJO: MAESTRO Y ESCLAVO

Diagrama de flujo del microcontrolador maestro:

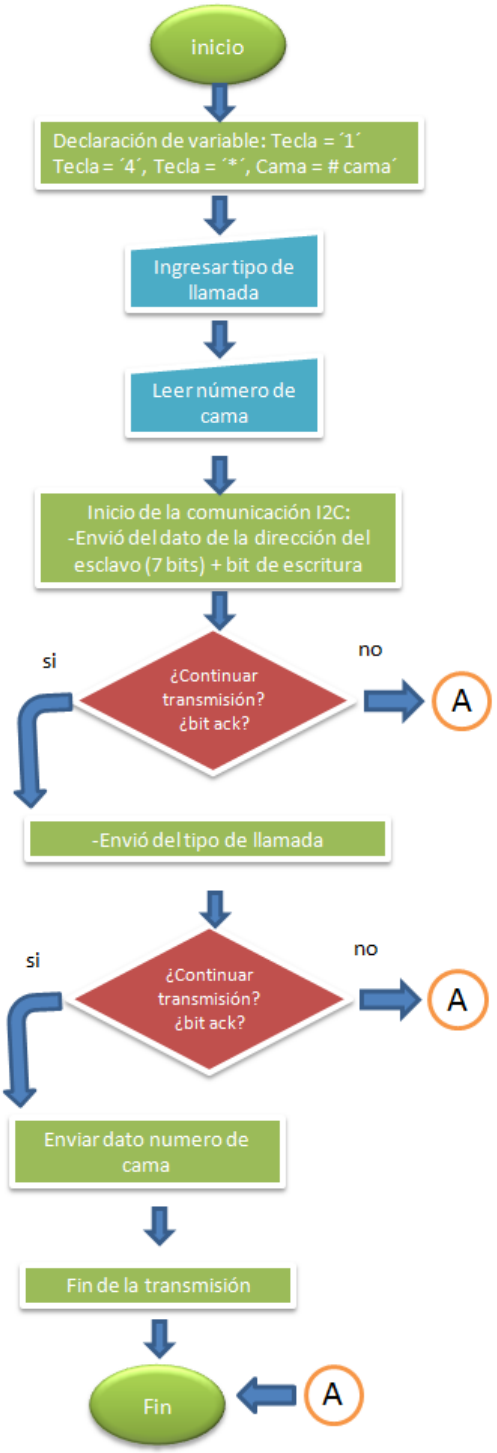
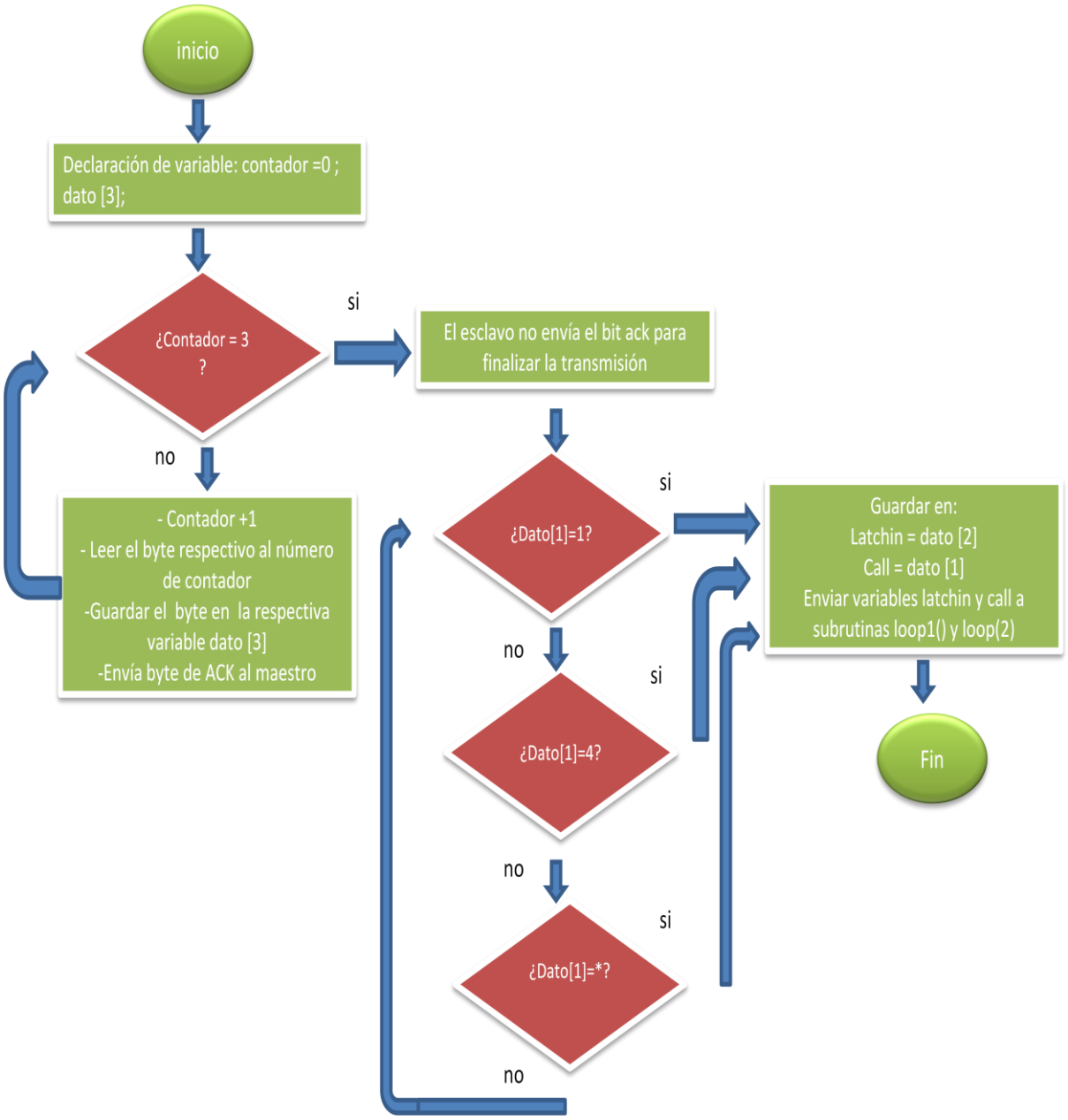


Diagrama de flujo del microcontrolador esclavo:



3.3 CÓDIGO FUENTE DEL MAESTRO

```
#include <16F819.h>
#fuses XT,PUT,NOPROTECT,BROWNOUT,NOLVP,NOWDT,NOMCLR
#use delay (clock=4000000)
#use i2c(MASTER, SDA=PIN_b0, SLOW, SCL=PIN_b1, NOFORCE_SW)
#include <kbda.c>
#use standard_io(b)
char tecla;
char cama;

/*****************************************************************************/
/***************************** Envío I2C *************************************/
void enviar_I2C (){

      i2c_start();
      i2c_write(0xa0);
      delay_us(1);
      i2c_write(tecla);
      delay_us(1);
      i2c_write(cama);
      delay_us(1);
      i2c_stop();
       }
/*****************************************************************************/
/***************************** FUNCIÓN PRINCIPAL *****************************/

void main(){
   port_b_pullups(TRUE);

   kbd_init();                      //Inicializa función de teclado
   set_tris_a(0xF8);                //configuracion del puerto a como i/o
   set_tris_b(0xFF);                //configuracion del puerto b como i/o

output_a(0);                        //inicializo las salidas de b a cero
      while (true){
           tecla=kbd_getc();        //En "tecla" valor de la tecla pulsada

     if(tecla!=0){                  //Si se pulsa teclado...

            if (tecla=='1'){
            (cama=input_b()>>2);
            enviar_I2C();
            output_high(PIN_a1);
            output_low(PIN_a2);
            delay_ms(1);
            }

            if (tecla=='4'){
            (cama=input_b()>>2);
            enviar_I2C();
            output_low(PIN_a1);
            output_high(PIN_a2);
            delay_ms(1);
            }

            if (tecla=='*'){
            (cama=input_b()>>2);
            enviar_I2C();
            output_low(PIN_a1);
            output_low(PIN_a2);
            delay_ms(1); }
      }
   }
}
```

COMENTARIO DEL CODIGO:

El microcontrolador maestro es un PIC16F819, que lleva conectado un oscilador de cuarzo de 4Mhz, con el fin de indicarle al microcontrolador la velocidad de trabajo. El oscilador genera una onda cuadrada de alta frecuencia que se utiliza como señal para sincronizar todas las operaciones del sistema. Es por eso que se especifica en #fuses que se va a ocupar un oscilador XT (cristal de cuarzo) ya que existen otro tipo de osciladores.

Los pines que se usan para el cristal también pueden utilizarse como puertos de entrada y salida de datos, por tanto existe la opción de utilizar el oscilador interno del PIC con la instrucción setup_oscillator (OSC_4MHZ).

El #FUSE PUT (power up timer) es un temporizador de encendido que al activarsegeneraunretrasoen el arranquedelmicrocontroladorantes de ejecutarcualquier código o de resetearse. #FUSE BROWNOUT para generar un reset cuando existe una baja de voltaje. NOLVP deshabilita el pin asignado para la programación de bajo voltajecon NOLVP el pin RB3/PGM se convierte en unpin de E / S digital.

Ahora bien, los microcontroladores pueden entrar en un estado en el que se detiene el oscilador del sistema, a esto se le conoce como modo Sleep del microcontrolador. Cuando esto sucede dejan de funcionar todas las partes del microcontrolador que dependen de él, es decir, se "congela" la ejecución del programa sin que los valores de los registros y puertos del microcontrolador sean alterados. La ventaja que ofrece para el proyecto el modo Sleep es que el microcontrolador solo trabajara cuando se presente una llamada de atención, y asíse ahorrara energía. Sin embargo el Watchdog timer (temporizador perro guardián) podría hacer que el microcontrolador saliera del modo Sleep al desbordarse. Es por tal razón que se utiliza el #FUSE NOWDT para desactivar al Watchdog. Finalmente el #FUSE NOMCLR desactiva el reset externo que puede conectarse a ese pin y lo configura como un pin de entrada o salida de datos, es importante este ultimo fuse ya que al oprimirse la tecla de '*' puede ocasionar que el microcontrolador quede pasmado.

Ya hemos visto que la directiva #use I2C, nos sirve para configurar las opciones para el protocolo I2C, configuramos al PIC16f819 como maestro y asignamos al Pin 0 del puerto B como la línea SDA y al pin 1 del puerto B como la línea SCL, la velocidad de transmisión de datos esta configura como la estándar que es la lenta y finalmente con NOFORCE_SW le decimos al micro que no forcé la comunicación vía software, es decir que las funciones I2C no se hagan por software sino por hardware utilizando los módulos SSP (para solo esclavo) ó MSSP (para esclavo o maestro).

Los dispositivos de llamada del paciente utilizan botones conectados a resistencias Pull-up para enviar la información, lo que se asimila a un teclado matricial. Por tal motivo decidí modificar el driver KBD.C del compilador de C que funciona para un teclado matricial de 3x4 para convertirlo en un teclado matricial de 1x3(una columna, tres filas) y que guarde con el nombre de KBDA.C, el nombre del archivo es debido a que también modifique el puerto de lectura de las teclas, en lugar de utilizar el puerto *b*, modifique en la

librería las líneas correspondientes para que se utilice el puerto *a* y queda de la siguiente manera:

```
23  #if defined(__PCH__)
24  #if defined use_porta_kbd
25     #byte kbd = 0xF80                    // This puts the entire structure
26  #else
27     #byte kbd = 0xF81                   // This puts the entire structure
28  #endif
29  #else
30  #if defined use_porta_kbd
31     #byte kbd = 5                     // on to port A (at address 5)
32  #else
33     #byte kbd = 6                    // on to port D (at address 8)
34  #endif
35  #endif
36  #if defined use_porta_kbd
37     #define set_tris_kbd(x) set_tris_a(x)
38  #else
39     #define set_tris_kbd(x) set_tris_b(x)
40  #endif
```

De esta manera puedo asignar a cada tecla un identificador propio, en este caso defino la tecla 1 para una llamada normal, la tecla 4 como una llamada de urgencia, y la tecla con el símbolo * para borrar ambas llamadas. A continuación se muestra un extracto del fichero KBDA.C donde aparecen los valores de las teclas una vez modificada la librería para un teclado matricial de 1x3.

```
68  // Keypad layout:
69  char  const KEYS[3][1] = {{'1'},
70                            {'4'},
71                            {'*'}};
72
73
74  #define KBD_DEBOUNCE_FACTOR 33     // Set this number to apx n/333 where
75                                     // n is the number of times you expect
76                                     // to call kbd_getc each second
77
```

Además, el usar esta librería y modificarla tiene otra ventaja como se observa a continuación. La imagen 3.3.a muestra el comportamiento de un interruptor al ser pulsado, éste presenta un efecto de rebote ó múltiples transiciones para una entrada única antes de alcanzar la estabilidad.

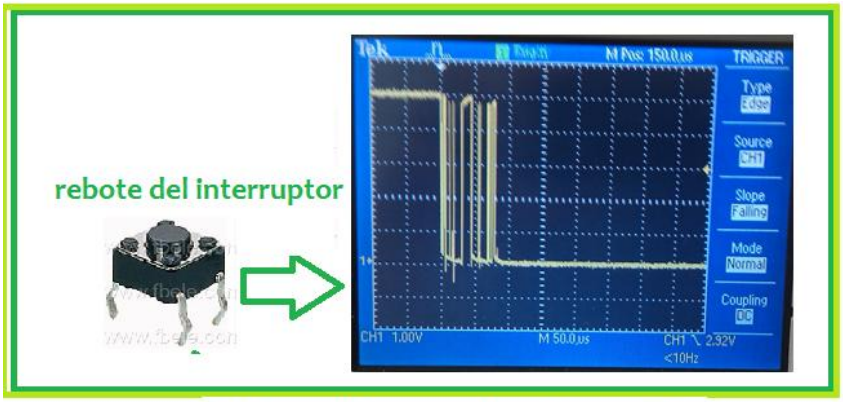


Fig. 3.3.a debounce de un interruptor

Debido a que el microcontrolador es demasiado rápido como para poder ver todas las transiciones puede interpretarlas como múltiples entradas. Sin embargo la librería KBDA.C considera este efecto y con la línea #define KBD_DEBOUNCE_FACTOR *n*, se minimiza el factor de rebote de los botones. Donde *n* es el número de veces que se espera llamar a *kbd_getc* cada segundo, por defecto tiene un valor de 33, sin embargo lo podemos disminuir para disminuir el número de lecturas y omitir los rebotes.

Para concluir el inicio de nuestro programa, definimos al puerto a y al puerto b como entradas y salidas estándar; y definimos nuestras variables a utilizar, donde las variables tecla y cama serán del tipo carácter.

Antes de comenzar con el programa principal se programa la subrutina que se encargara de desarrollar el protocolo I2C y que a continuación aparece con los comentarios correspondientes:

```
13 /*******************************************************************************/
14 /****************************** Envío I2C **************************************/
15
16 void enviar_I2C (){
17
18       i2c_start();          //Comienzo de la comunicación I2C ...
19       i2c_write(0xa0);      //...con la dirección del PIC esclavo...
20       delay_us(1);
21       i2c_write(tecla);     // Envia dato que indica el tipo de llamada
22       delay_us(1);
23       i2c_write(cama);     // Envia dato que indica el numero de cama
24       delay_us(1);
25       i2c_stop();           //Finalización de la transmisión
26        }
27 /*******************************************************************************/
```

Una vez definida la función que establece la comunicación, se programara la función principal.

Primero definimos las condiciones iníciales del programa. En el puerto B, a partir del pin 2 estará conectado un dispositivo dip-switch de seis canales con el que podremos configurar hasta un valor de 63 camas de paciente. Para poder utilizarlo necesitamos tener resistencias pull-up conectadas al dispositivo, sin embargo este puerto nos ofrece lo opción de activar las resistencias pull-up internas del microcontrolador con la instrucción: *port_b_pullups(TRUE);*Activar estas resistencias por software no solo resulta útil para el dip-switch sino que además son útiles para el módulo I2C. De ahí que al iniciar la simulación en PROTEUS RB0, RB1, RB2, RB4 aparecen en color rojo para indicar que están conectadas a una tensión mediante Pull-ups.

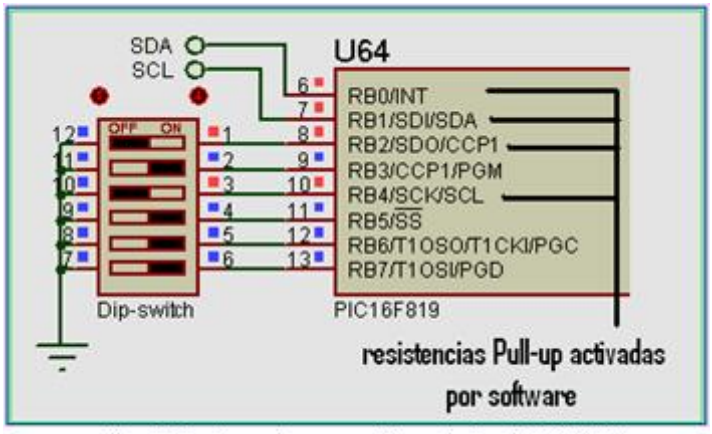


Fig. 3.3.b Conexion entre dip-switch y PIC16f819

La función kbd_init ( )que incorpora el driver KBDA.c inicializa el sistema para manejo el del teclado y esta debe ser la primera función en el programa. Puesto que todo el *puerto b* va a ser utilizado como puerto de entrada de datos utilizamos la función set_tris_b(0xFF), donde los 1 indican entradas y los 0 salidas.

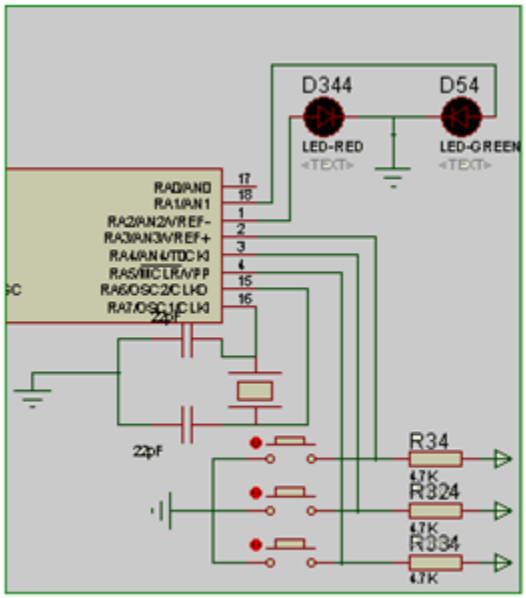


Fig. 3.3.c Conexion dispsitivos de salida y entrada en puerto A

A continuación configuramos el *puerto a;* desde el pin *A0* hasta el pin *A2*se utilizaran como salidas, estos pines se utilizaran para encender un par de leds, A partir del pin A3 hasta el pin A5 los utilizaremos como entradas, donde estarán los botones que indican el tipo de llamada, para lograr esta configuración utilizamos la función *set_tris_a(0xF8)*. Los pines A6 y A7 serán utilizados para conectar un cristal de cuarzo a 4Mhz puesto que esos pines ya habían sido configurados con el #fuses XT.

Comenzamos ahora la función principal, el bloque de sentencias dentro de la sentencia *while (true)*se ejecutara de manera repetitiva, puesto que se evalúa como verdadera la expresión lógica dentro del paréntesis. El bloque dentro de la sentencia *while* se denomina bucle y cada ejecución se denomina iteración.

La primera iteración *tecla=kbd_getc()*devuelve un valor de acuerdo a la tabla programada en la librería KBDA.c Este valor es asignado a una variable denominada *tecla.*

Mientras no sea oprimido ningún botón del *puerto a* no sucederá nada, en el momento en que el valor de la *tecla* sea diferente de cero, se evaluaran tres diferentes situaciones: si la tecla es “1”, si la tecla es “4” y si la tecla es “*”. Para los tres casos el procedimiento es el mismo, lo único que los diferencia es que se prende un led verde ó bien un led rojo, ó ambos se apagan.

La expresión *cama=input_b()>>2* es de suma importancia ya que si no se define a partir de que pin del *puerto b* obtendremos el valor que se le asignará a la variable *cama*, se corre el riesgo de un error. Con la condición *input_b ()>>2* se le dice al microcontrolador que el valor se leerá a partir del pin b2, omitiendo de esta manera los pines asignados a SDA y SCL. Finalmente los valores de *tecla* y *cama* se envían por medio de la subrutina *enviar_I2C();* al microcontrolador esclavo.

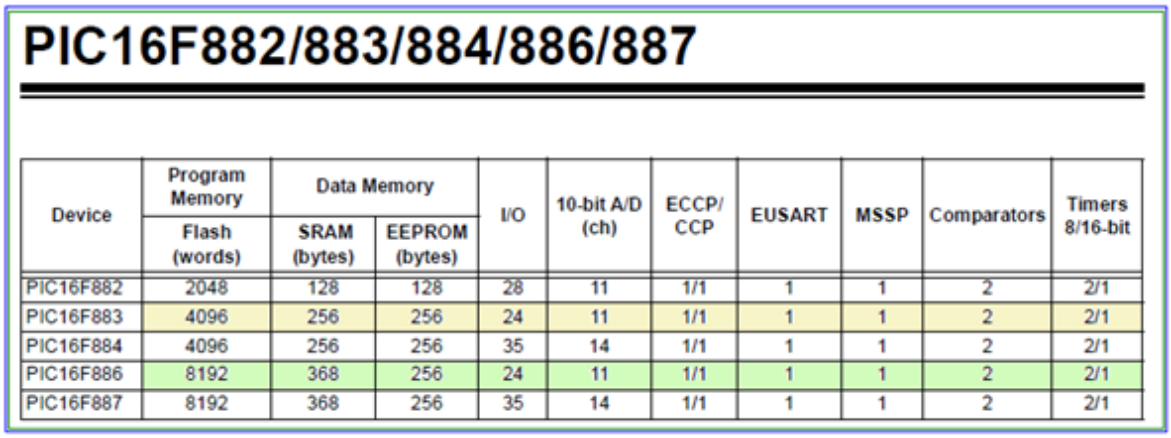
3.4 CÓDIGO FUENTE DEL ESCLAVO

```
#include <16f886.h>
#fuses XT,PUT,NOPROTECT,BROWNOUT,NOLVP,NOWDT
#use delay (clock=4000000)
#use fast_io(C)
#use standard_io(A)
#use standard_io(B)
#include <latchmem3.c>
#use I2C(SLAVE, SDA=PIN_C4 ,SLOW, SCL=PIN_C3, ADDRESS=0xa0, NOFORCE_SW)
int dato[3] ;
int count;

void main() {
set_tris_c(0x18);
output_c(0);
   while (TRUE) {
// Recepción por comunicación I2C
//i2c_poll solo se puede usar por hardware, detecta un byte en el buffer
//seguidamente se llama a la funcion i2c_read que devolvera el byte recibido
count=0;
while(count!=3)
{
while(!i2c_poll()) ;
{
dato[count++]= i2c_read(); //Lee el siguiente dato
}
}
        if (dato[1]=='1'){
           latchin=dato[2];
           call=dato[1];
           loop1();
           loop2();
           }
        if (dato[1]=='4'){
           latchin=dato[2];
           call=dato[1];
           loop1();
           loop2();
           }
        if (dato[1]=='*'){
           latchin=dato[2];
           call=dato[1];
           loop1();
           loop2();
           }
}
}
```

COMENTARIO DEL CODIGO:

Originalmente se había elegido al PIC16f883, sin embargo la memoria ROM se desbordaba y no nos permitía compilar el programa, con ayuda del buscador del siguiente link: http://www.microchip.com/maps/Microcontroller.aspxseencontro un PIC con las mismas características físicas pero con mayor capacidad de memoria ROM, este fue el PIC16f886. En la siguiente tabla de comparaciones vemos que la memoria flash aumenta al doble.



PIC16F882/883/884/886/887

Device	Program Memory	Data Memory		I/O	10-bit A/D (ch)	ECCP/ CCP	EUSART	MSSP	Comparators	Timers 8/16-bit
	Flash (words)	SRAM (bytes)	EEPROM (bytes)							
PIC16F882	2048	128	128	28	11	1/1	1	1	2	2/1
PIC16F883	4096	256	256	24	11	1/1	1	1	2	2/1
PIC16F884	4096	256	256	35	14	1/1	1	1	2	2/1
PIC16F886	8192	368	256	24	11	1/1	1	1	2	2/1
PIC16F887	8192	368	256	35	14	1/1	1	1	2	2/1

Fig. 3.4.a cuadro comparativo entre distintos pics de la misma familia

Los #FUSES que utilizamos son prácticamente los mismos que para el código del maestro a excepción del #FUSE NOMCLR que se omite. Debido a que ya se explicócuál es la función de cada uno, se omitirán los comentarios respectivos. El #FUSE NOMCLR se omite debido a que ese pin no se utiliza como salida o entrada de algún dato y además porque en el circuito impreso si es necesario tener un botón de *reset* para cualquier anomalía que se presente durante su funcionamiento.

La librería *latchmem3.c*se explicara en el capítulo que trata el tema de la implementación de los *latches*, solo basta decir que esta librería es la que nos ayuda a encender o apagar varios leds de forma independiente de nuestro panel del control de enfermería.

Nuevamente usaremos la directiva #use I2C, pero ahora para configurar al PIC como esclavo para eso usamos la instrucción SLAVE dentro de los paréntesis, el pin C4 se utilizara para la línea SDA y el pin C3 para la línea SCL, la velocidad de transmisión es la baja y la dirección que le asignamos a este dispositivo es la 0xa0, la cual debe de coincidir con la direccionada por los dispositivos maestros. Finalmente este PIC cuenta con el módulo MSSP según su hoja de características y por tanto no requerimos la implementación del protocolo I2C por software (NOFORCE_SW).

Son dos las variables las que se declaran, *count* que es del tipo entero y que tiene como función ser un contador que se incrementara cada que se reciba una dato mientras se cumpla la condición de *count!=3*. La variable *intdato[3]*es un array de caracteres; recordemos que un array de caracteres es un conjunto de variables del mismo tipo de datos, cada una de esas variables se coloca en forma consecutiva en la memoria RAM del PIC y se les conoce como elementos del array. Los elementos del array se enumeran empezando por el 0(es una característica del lenguaje C).

En este caso *intdato[3]*está declarando un array de caracteres del tipo entero que puede almacenar como máximo 3 elementos.

Dentro de los paréntesis de la siguiente sentencia *while*, se encuentra la función *!I2C_poll()*, en un capitulo anterior ya se había descrito su modo de operación. Solo hay

que recordar que esta función implementada por hardware, detecta un byte en el buffer y regresar un valor TRUE, acto seguido llama a la función*I2C_read* para devolver el byte recibido.

El valor devuelto por la función *I2C_read* se almacena en el primer elemento del array para posteriormente incrementarse y así almacenar las tres variables enviadas por el circuito maestro.

Finalmente tenemos tres condiciones *IF*, que mandan a las subrutinas *loop1()* y *loop2()* los valores de *dato[1]* y *dato[2]* para ser procesados y establecer las condiciones para operar los latches.

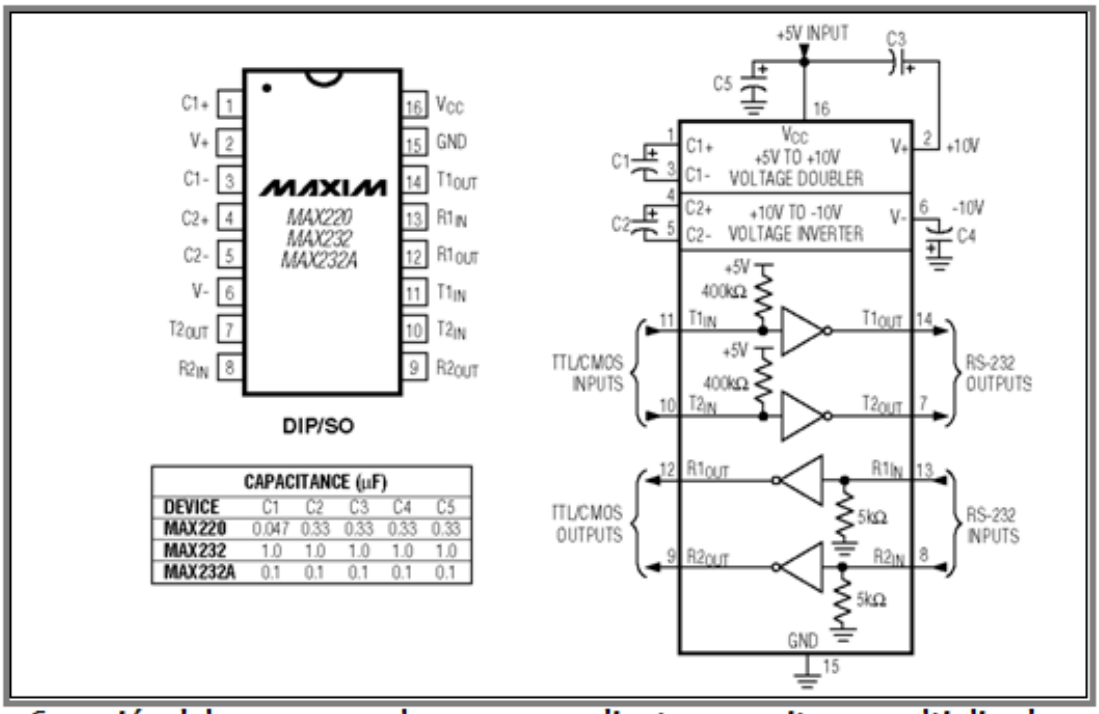
CAPITULO 4: TRANSMISIÓN REMOTA

4.1. INTRODUCCIÓN

Como se ha mencionado el protocolo I2C solo resulta funcional para distancias cortas. Las primeras pruebas realizadas eran a distancias menores a los tres metros y funcionaban perfectamente, el problema surge cuando se utiliza un cable con una longitud de cien metros ya que el circuito esclavo dejaba de responder. La solución podría encontrase en un circuito capaz de aumentar la distancia de comunicación de I2C, este circuito ya existe en el mercado con la el nombre de PCA 9600, sin embargo utilizarlo haría a un lado el objetivo de utilizar componentes comunes y fáciles de conseguir. De tal forma que se pensó en utilizar el circuito MAX232.

4.2. C.I.MAX 232

El MAX232 es un circuito integrado de Maxim que convierte las señales de un puerto serie RS-232 (hasta de ± 30 V), a señales compatibles con los niveles TTL de circuitos lógicos de + 5 V, y viceversa, con ayuda de multiplicadores de voltaje (etapas de diodos y capacitores)adicionando al circuito condensadores externos. Los multiplicadores de voltaje resultan de gran ayuda para compensar la caída de tensión debido a la longitud de las líneas.



CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	0.047	0.33	0.33	0.33	0.33
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Fig. 4.2.a Conexión del max 232 con los correspondientes capacitores multiplicadores de voltaje

El funcionamiento es el siguiente, si el MAX232 recibe un nivel lógico TTL de 0,lo cambiara por un voltaje comprendido entre +3 y +15 V, y cuando recibe un nivel lógico TTL de 1 lo cambia por un voltaje comprendido entre -3 a -15 V, y viceversa, para convertir niveles de RS232 a TTL.

El circuito entonces sirve como interfaz de transmisión y recepción para las señales RX, TX.Donde la primera señal es la de recepción de datos y la segunda señal es la de transmisión de datos.

La versión MAX232A es compatible con la original MAX232, y tiene la mejora de trabajar con mayores velocidades de transferencia de información (alcanza hasta 120 kbit / s), lo que reduce el tamaño de los condensadores externos utilizados para el multiplicador de voltaje, de esta forma podemos utilizar un capacitor de 0.1 µF en lugar de uno de 1.0 µF.

El circuito mostrado en la figura 4.2.b utiliza una fuente de 12 Volts a 2A, y cada dispositivo tiene un regulador de voltaje a 5 Volts. Las primeras pruebas se realizaron utilizando una fuente de 5 Volts a 800mA pero por la distancia entre las líneas había una caída de corriente considerable al punto de que no se lograba la intercomunicación entre los dispositivos maestros y esclavo.

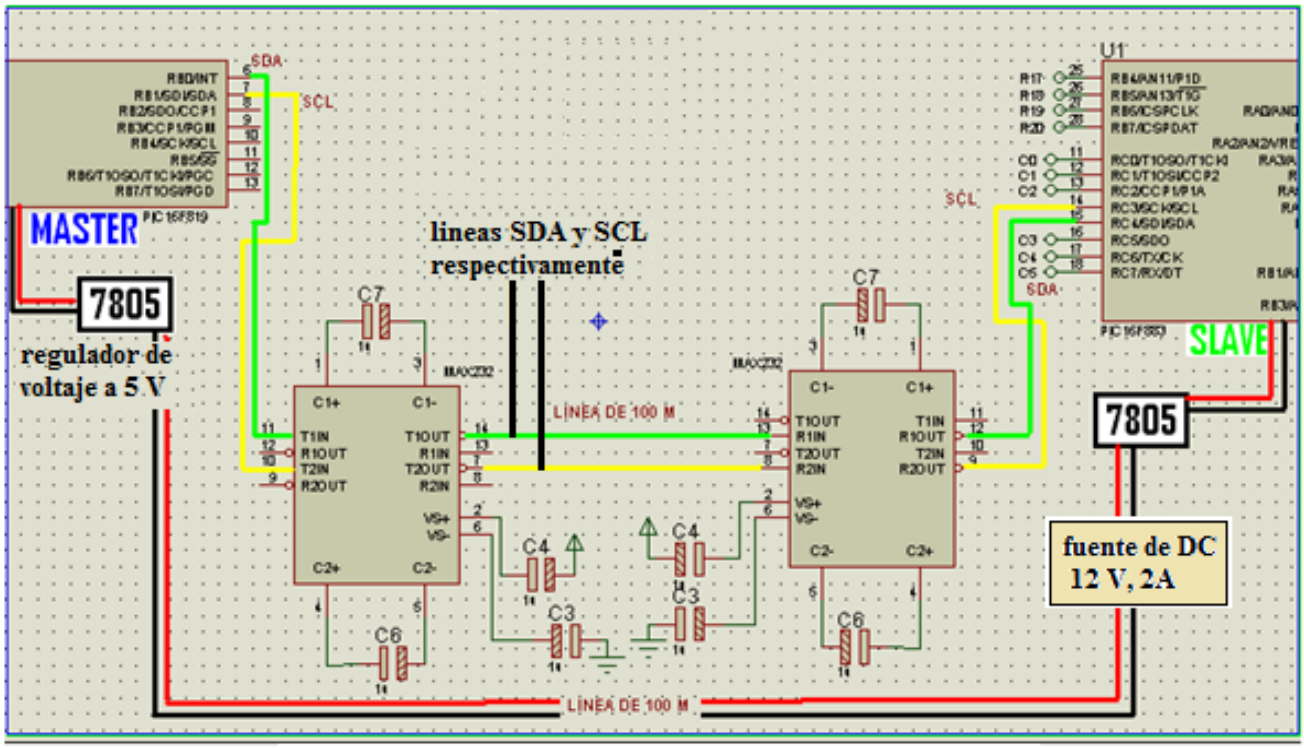


Fig. 4.2.b Diagrama de conexiones entre dispositivos esclavo y maestro

4.3. PRUEBAS

Las pruebas se realizan bajo tres condiciones de operación distintas y se analizan con ayuda de un osciloscopio digital.

La primera prueba se realiza sin utilizar el MAX232 y con una línea menor de tres metros,en este caso los trenes de pulsos de SCL y SDA tanto a la salida del circuito maestro como a la entrada del circuito esclavo son los mismos, es decir, que durante la transmisión de datos no se experimenta ninguna atenuación o deformación de las señales.

La segunda prueba se realiza utilizando el MAX232 y con una línea cuya longitud es de tres metros. Las figuras 4.3.a y 4.3.b muestran las señales SDA y SCL respectivamente, que genera el dispositivo maestro para comunicarse con el dispositivo esclavo.

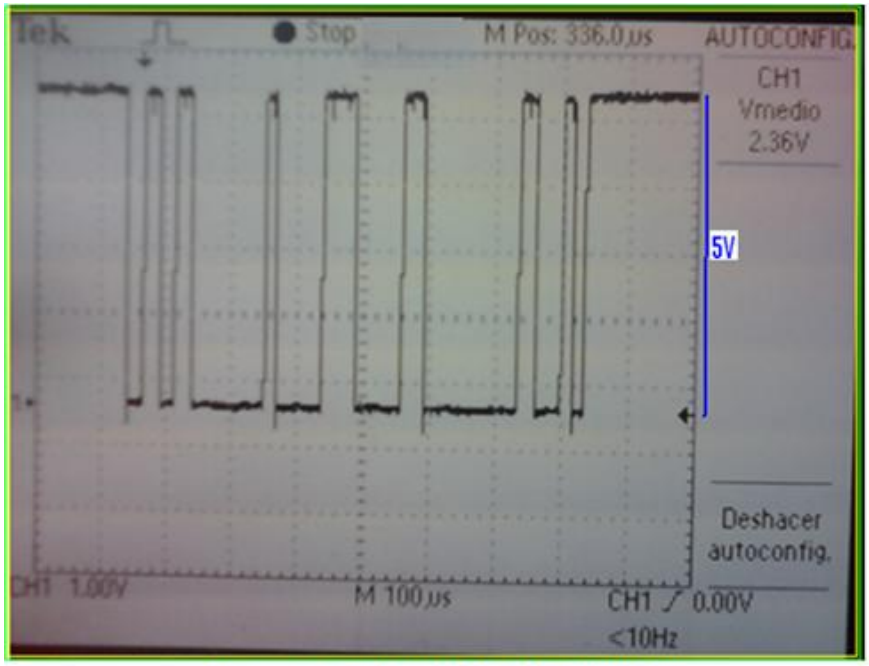


Fig. 4.3.a Señal SDA al salir del PIC16f819

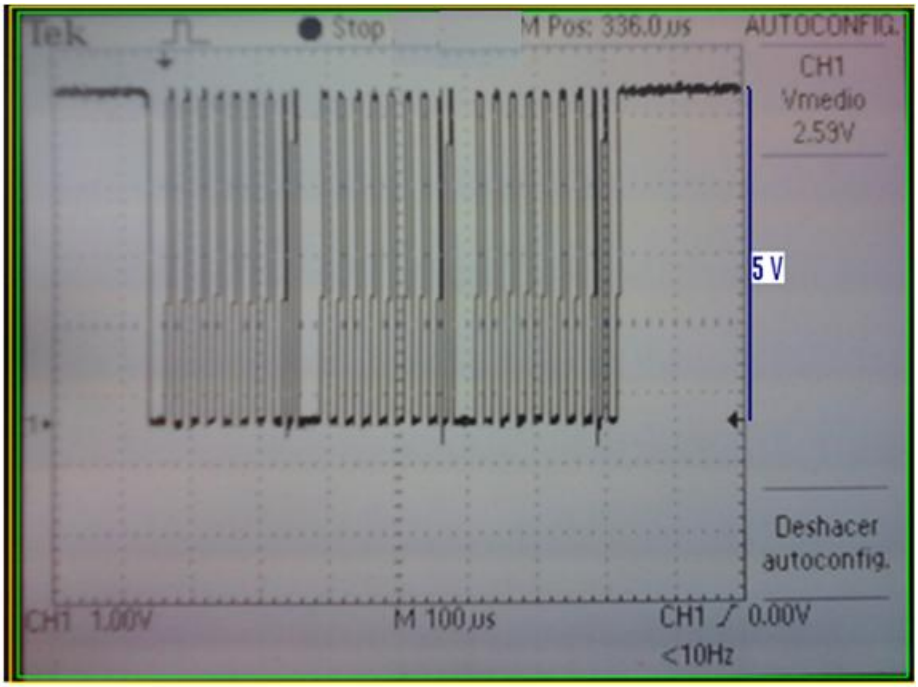


Fig. 4.3.b Señal SCL al salir del PIC16f819

Dichas señales entran por el MAX 232, con niveles de voltaje de 0 V a 5 V, posteriormente estas señales salen del MAX232con niveles de voltaje de -3V a 3 V, como se observa en las figuras 4.3.c y 4.3.d

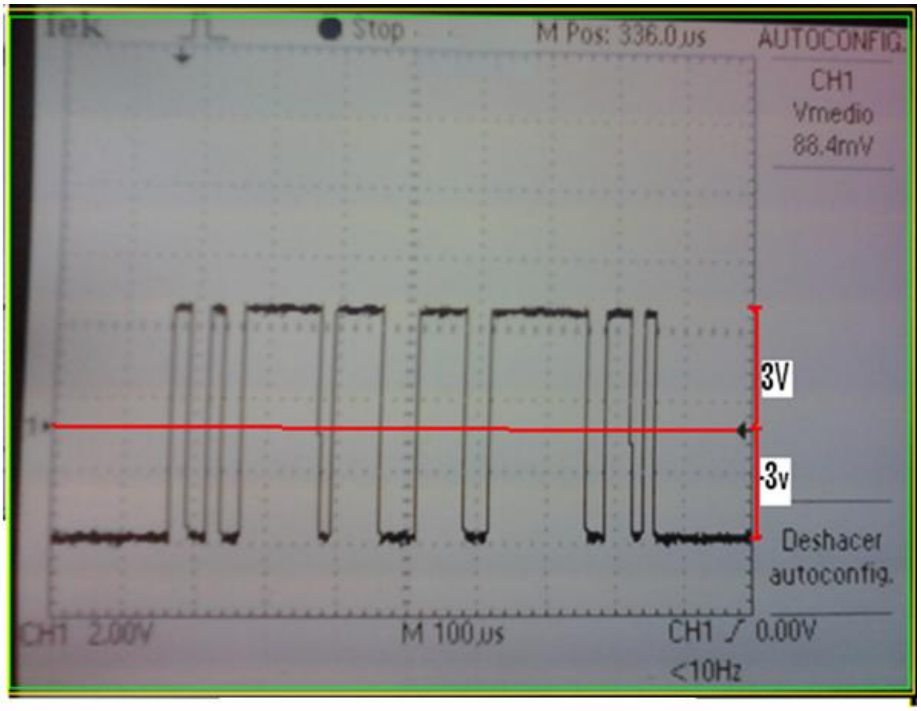


Fig. 4.3.c Señal SDA en nivel de voltaje RS232

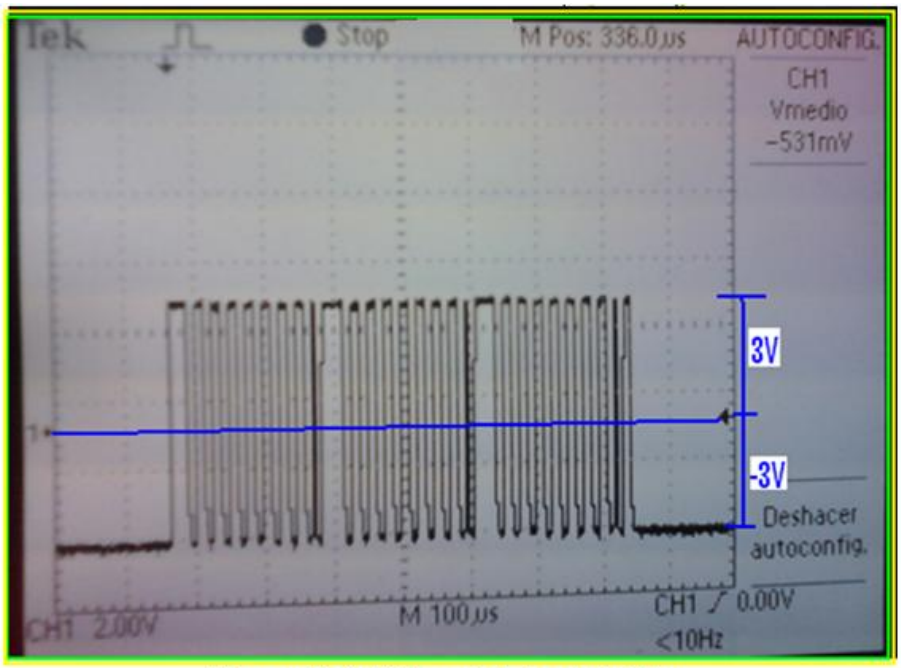


Fig. 4.3.d Señal SCL en nivel de voltaje RS232

Posteriormente estas señales atraviesan la línea y llegan al MAX 232receptor. Las señales que se visualizan en el osciloscopio son las mismas que de las figuras 4.3.c y 4.3.d

Finalmente, una vez convertidos los voltajes al pasar por el MAX232receptor se observa que las señales que van hacia el PIC son los mismos trenes de pulso enviados por el dispositivo maestro visualizados en las figuras 4.3.a y 4.3.b, con apenas algo de ruido para SDA y casi nula para SCL

Hasta este momento no se ha presentado algún problema en la transmisión y recepción de datos.

Ahora bien, la tercera condición de prueba y la que nos interesa en este capítulo es aquella que involucra la línea de cien metros la que determinara si en realidad nos es útil o no el circuito MAX232.

Como punto de partida se analiza el comportamiento de la señal SDA al ser transmitida sobre una línea de longitud de cien metros y sin utilizar el MAX 232, el resultado es el mostrado en la Fig.4.3.e

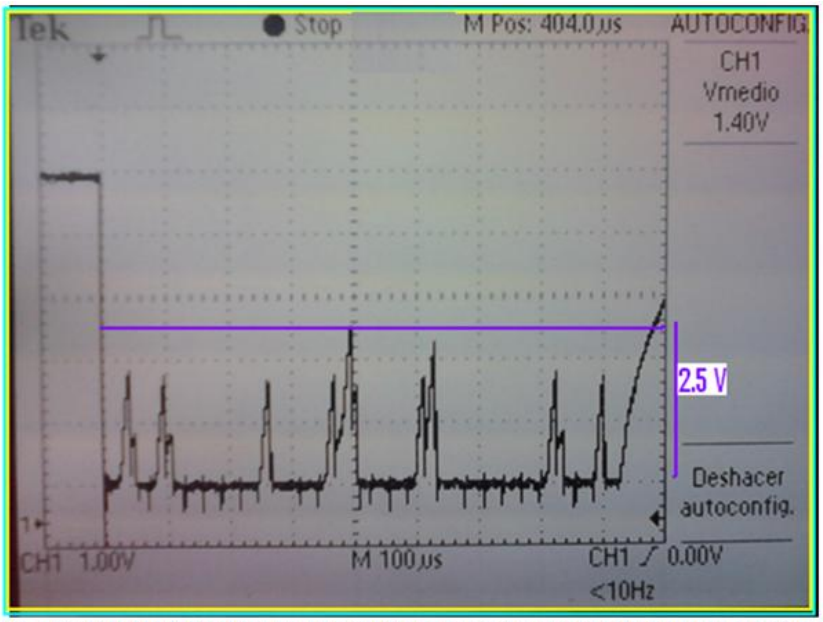


Fig. 4.3.e Señal SDA del PIC16f819, prueba con 100 metros de cable teléfonico sin el max 232

La imagen muestra que al inicio, antes de realizar la llamada, el voltaje es de 5 V debido a las resistencias pull-up; Una vez realizada la llamada, el flanco de bajada indica el START de la transmisión, sin embargo el tren de pulsos sufre una atenuación considerable que hace que el circuito esclavo sea incapaz de interpretar la información.

Al implementar el circuito de la fig. 4.2.b.el cual ya incluye al MAX232, resulta que las señales SDA Y SCL no sufren ninguna atenuación al salir del dispositivo maestro como en la prueba anterior. Sin embargo el problema viene después de que estas señales pasan por el MAX233, como se aprecia en la fig. 4.3.g.

Fig. 4.3.f señales SDA y SCL al salir del pic16f819

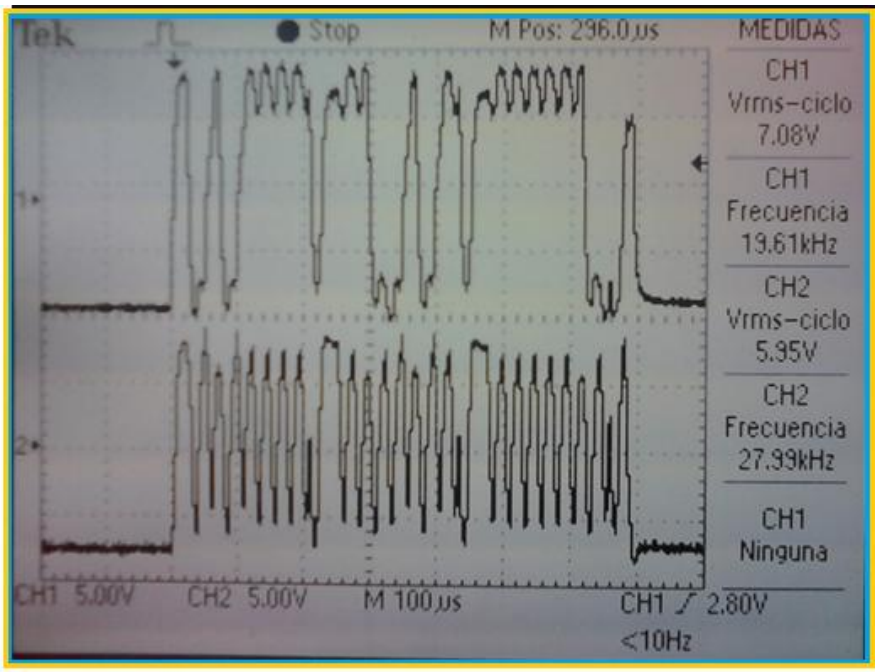


fig. 4.3.g. Señales SDA y SCL al salir del MAX232 del circuito maestro

Las señales mostradas en el osciloscopio presentan interferencia, aun así es posible la comunicación y el panel de leds responde al tipo de llamadas que se le asigna, sin embargo al hacer varias llamadas continuas y aleatorias llega un momento en el cual el PIC esclavo queda pasmado y necesita un reset. Este comportamiento se le atribuye al ruido que presentan las señales ya que quizás el PIC esclavo detecta falsos pulsos o en su defecto no detecta los niveles de tensión por debajo de lo establecido.

El objetivo ahora será eliminar estas interferencias y dejar las señales lo más estable posible, para tal fin se analizara de donde proviene dicho ruido.

Lo primero que debe ser contemplado son todos los parámetros que aparecen en las líneas de transmisión, en el sub-capítulo 2.2 se había mencionado que la capacitancia entre líneas hacia que la señal SDA sufriera una deformación, sin embargo esto es solo un parámetro a considerar y se presentaen un caso "estándar" de transmisión.

Pero ya que la línea de transmisión es de cien metros, los parámetros a considerar son aun más, como se muestra en la Fig. 4.3.h Estos parámetros hacen alusión al concepto de impedancia característica de línea.

La impedancia característica de una línea de transmisión se define como la relación existente entre la diferencia de potencial aplicado y la corriente absorbida por la línea ($z_0 = {}^{V}/_{I}$) y esta depende de: la resistencia, capacitancia, inductancia y conductancia (inversa de la resistencia de aislamiento entre los conductores que forman la línea), además de la frecuencia y geometría de la línea.La resistencia y la inductancia ocurren a lo largo de la línea, mientras que entre los dos conductores ocurren la capacitancia y la conductancia.

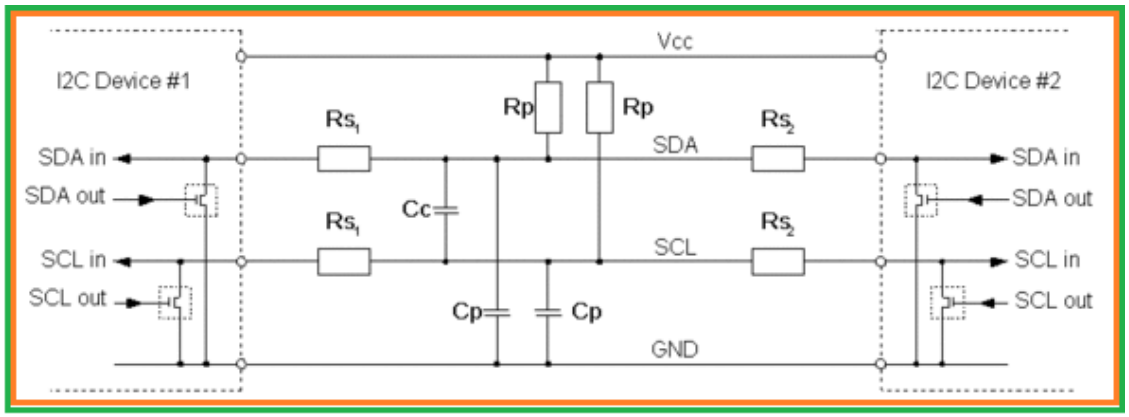


Fig. 4.3.h Diagrama del circuito equivalente para una conexion I2C

- Rs Impedancia en serie $Z = R+j\omega L$
- Cp es la capacitancia que se genera entre la líneas SDA y tierra y SCL y tierra
- Cccapacitancia cruzada entre SDA y SCL.

Lo importante es conocer cómo influyen estos parámetros sobre las señales.

El parámetro de capacitancia entre líneas, propicia pequeños picos en las señales. En adición a estos picos, aparece el fenómeno de diafonía (*Crosstalk*) que es el más notorio en la fig. 4.3.g. éste fenómeno propicia que una de las señales (SDA o SCL) se cuele de un cable a otro y esta es la razón de que los picos de la señal SDA aparentemente sean un reflejo de los pulsos de las señal de reloj.

Este fenómeno además de aparecer debido al efecto capacitivo e inductivo entre todos los hilos involucrados en el sistema de transmisión, también es consecuencia del parámetro de admitanciaya quedebido a la imperfecciónde sistema de aislación las corrientes de fugafluyen a través de las superficies de los aisladores influyendo un conductor sobre otro. Otra característica que debe ser considerada, es que la línea de 4 hilos constituye una antena simple y en ellos se pueden inducir fácilmente tensiones.

Una posible solución para despreciar el fenómeno de diafonía es el uso de cable UTP el cual lleva varios pares de hilos trenzados, y de los cuales podemos utilizar solo dos pares. Al estar trenzados los pares de hilos se reduce la interferencia eléctrica con respecto de los cables cercanos que se en encuentren alrededor debido a que las tensiones provenientes de ambos pares se contrarrestan. Sin embargo se opta por otra solución debido a que seguirán presentes los parámetros de la impedancia característica sobre la línea

La otra solución se encuentra enel diseño deun par de filtros paso-bajas activos de primer orden cuya frecuencia de corte será de 100khz para la señal de SDA y 120Khz para SCL, de tal forma que al diagrama de la Fig. 4.2se le adicionan un par de amplificadores operacionales (TL082) para filtrar SDA y SCL.

Las señales que salen de los filtros llegan a otros TL082 en configuración seguidores de voltaje para lograr el acoplamiento delas señales con el MAX232, ya que sin

estos últimos amplificadores, aun cuando se observa un buen nivel de tensión en el osciloscopio, no se da la comunicación con el PIC esclavo (Fig. 4.3.i)

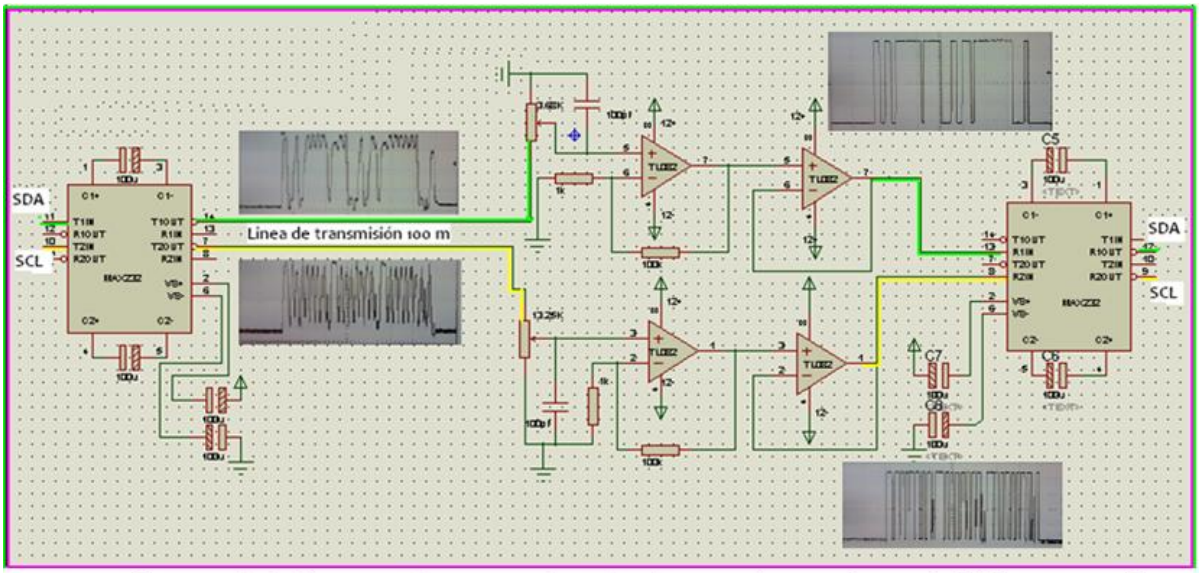


Fig. 4.3.i implementación de filtros paso- bajas y seguidores de voltaje con el TL082 a la entrada del dispositivo esclavo

Para el diseño de los filtros se utilizó el software FilterPro Desktop de Texas Instruments.

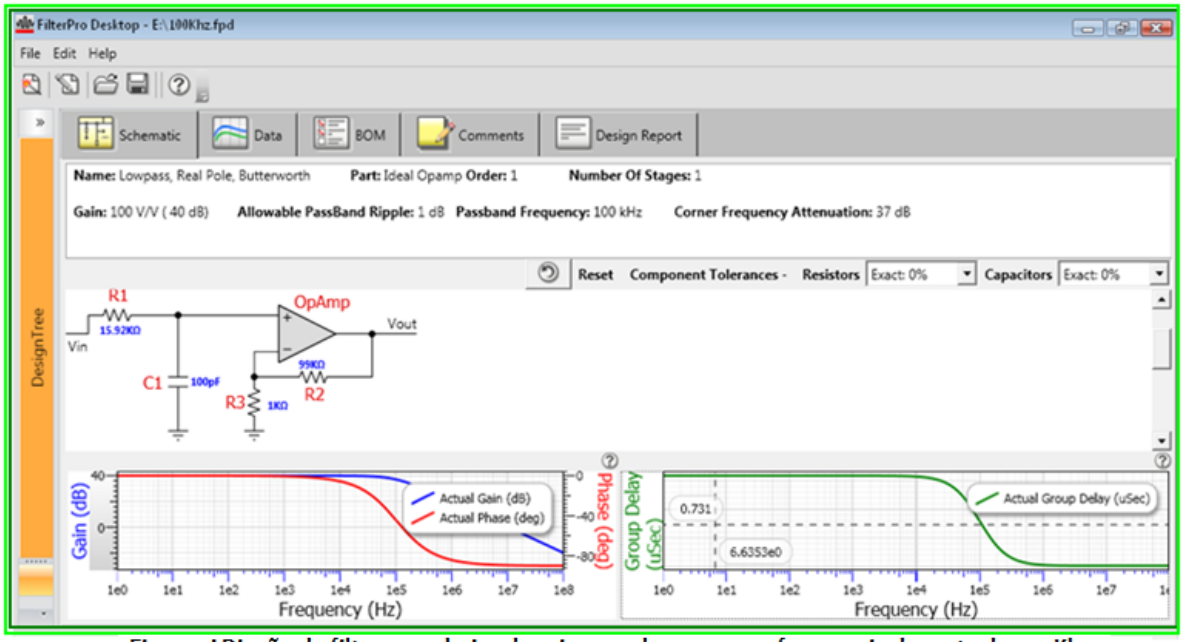


Fig. 4.3.j Diseño de filtro paso-bajas de primer orden para una frecuencia de corte de 100 Khz

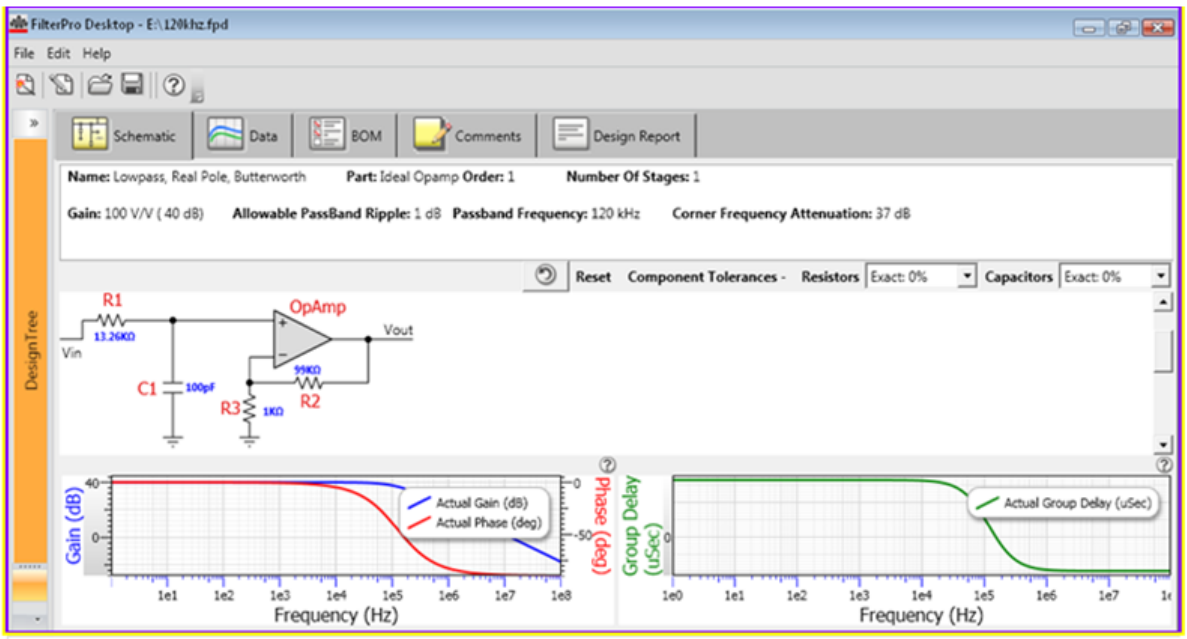


Fig. 4.3.k Diseño de filtro paso-bajas de primer orden para una frecuencia de corte de 120 Khz

De esta forma las señales SDA y SCL (Figuras 4.3.l y 4.3.m) quedan filtradas y listas para entrar al MAX232 y hacer la conversión a niveles TTL.

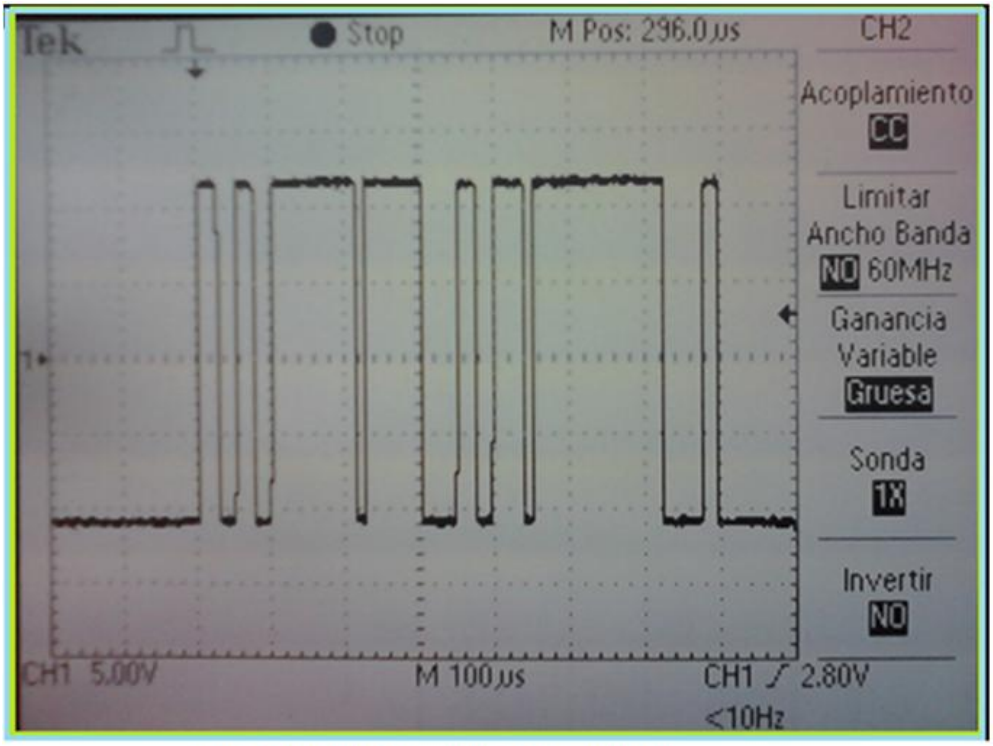


Fig. 4.3.l Señal SDA al pasar por el filtro paso-bajas

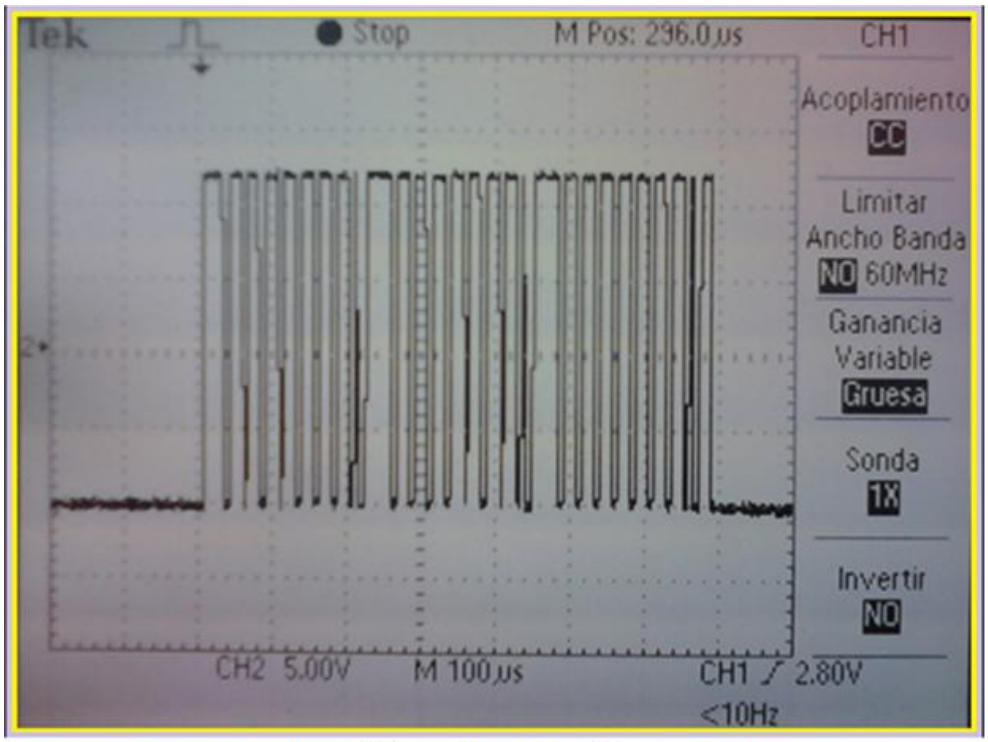


Fig. 4.3.m Señal SCL al pasar por el filtro paso - bajas

La Figura 4.3.n. muestra las señales SDA y SCL, despues de llegar filtradas al MAX232

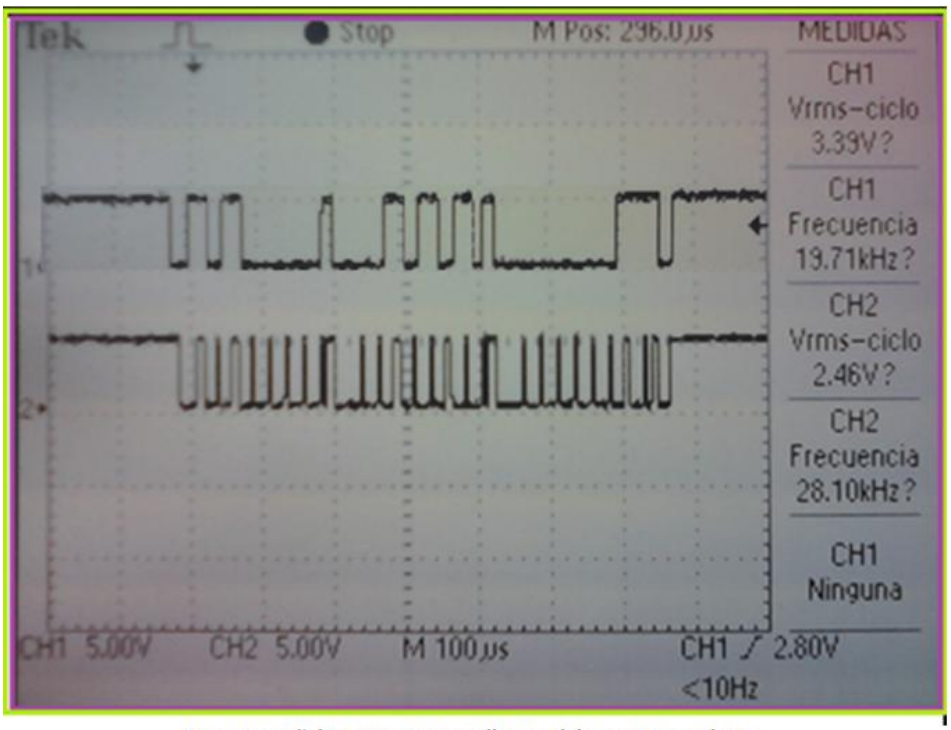


Fig. 4.3.n Señales SDA y SCL que llegan al dispositivo esclavo

Si se hace una comparación de cómo se observan las señales SDA y SCL cuando son enviadas desde el maestro y despues de ser procesadas hasta llegar al esclavo se observa que no muestran diferencia (Figura 4.3.o. y 4.3.p.) De esta forma se cumple el objetivo de la transmisión de datos por I2C a una distancia de cien metros.

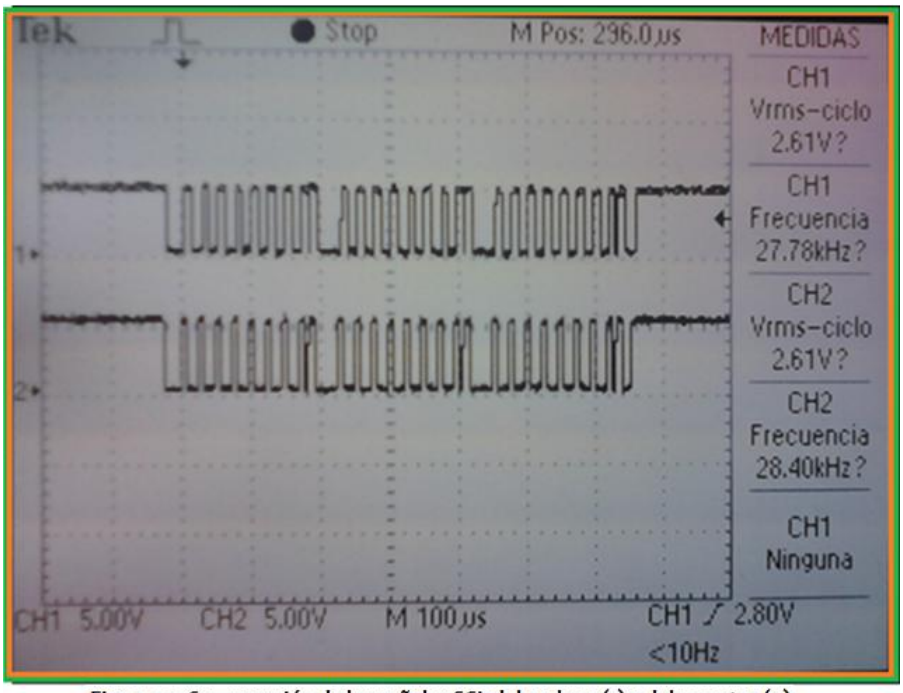


Fig. 4.3.o. Comparación de las señales SCL del esclavo (1) y del maestro (2)

Fig. 4.3.p Comparación de las señales SDA del esclavo (1) y el maestro (2)

CAPITULO 5: IMPLEMENTACION DE LATCHES

5.1 INTRODUCCIÓN

Hasta este momento se ha visto cómo hacer la transmisión de datos del dispositivo maestro al esclavo utilizando el protocolo I2C. Se ha solucionado el problema de la transmisión de datos cuando se hace a través de una línea con una longitud de cien metros con ayuda de un circuito multiplicador de voltaje como lo es el MAX232.

En el siguiente capítulo veremos cómo realizar la representación de los datos de forma visual, en este caso como se representaran el ó los números de cama que deseen recibir atención del servicio de enfermería. Para esto me apoyare de un panel de acrílico sobre el cual se han colocado 16 pares de leds (uno verde y otro rojo) que representaran el número de cama y el tipo de llamada.

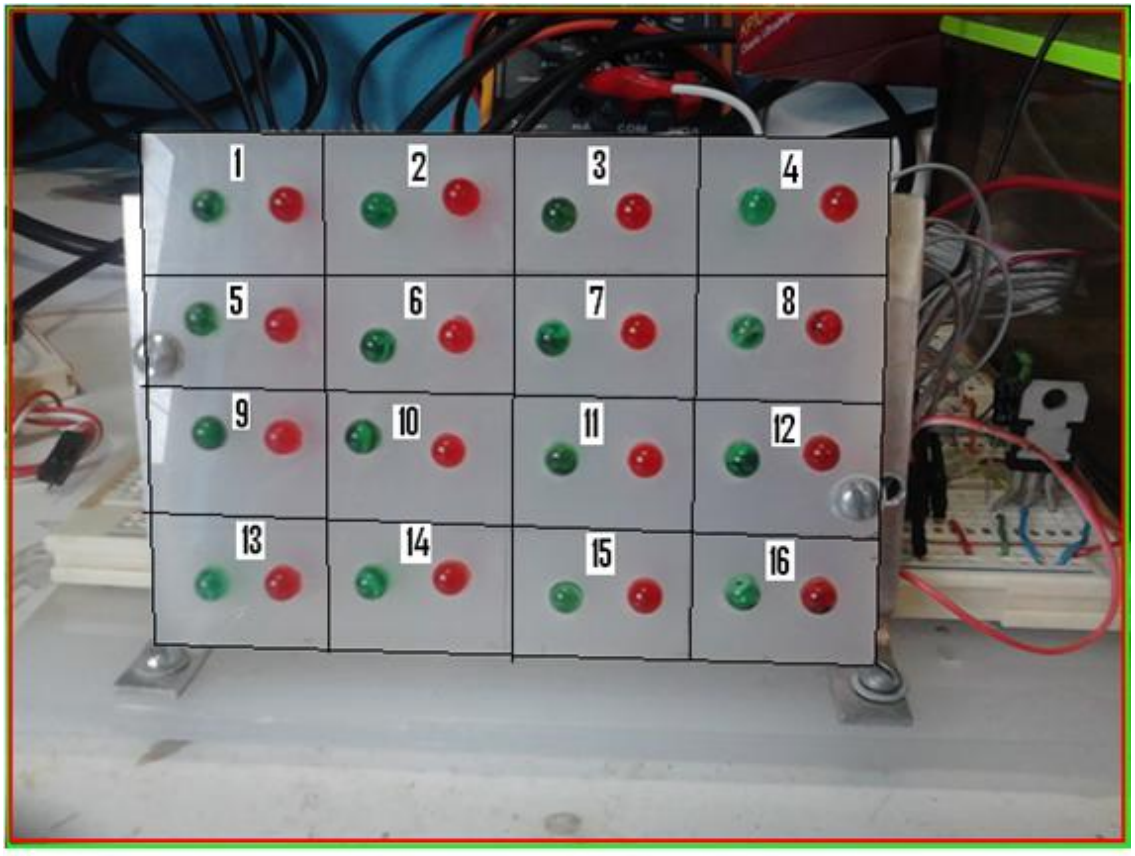


Fig. 5.1.a Prototipo módulo de control de enfermería con indicadores led

Los problemas a resolver en este capítulo son:

- ¿De qué manera utilizamos el mínimo número de pines del microcontrolador para controlar 32 leds?

- ¿De qué manera controlamos 32 leds de manera independiente, es decir sin afectar el estado anterior de cualquiera de ellos al ejecutar una llamada diferente?

La solución a estas incógnitas es un circuito latch, con el que podremos controlar ocho salidas de forma independiente, además este circuito cuenta con un pin de activación o desactivación para poder controlar varios latches a la vez. Lo cual nos será muy útil porque de esta forma con un solo puerto del microprocesador podremos controlar cualquier cantidad de leds en proporción al número de pines disponibles para activar varios latches.

El PIC16f886 que utilizamos, nos da la posibilidad de controlar 12 latches, 6 latches para controlar hasta 48 leds verdes y 6 latches más para controlar 48 leds rojos según el diagrama siguiente:

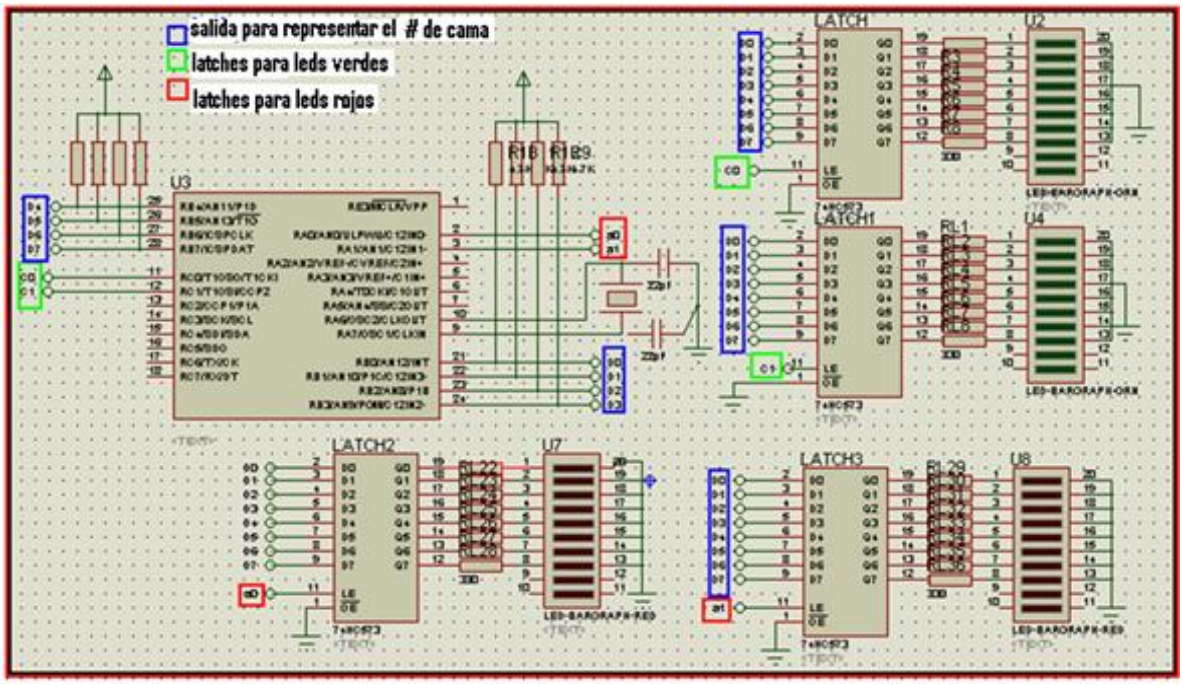


Fig. 5.1.b Diagrama de conexión entre el pic16f886 y los latches

El diagrama anterior por cuestiones de espacio en el simulador, únicamente está representando 2 latches para los leds verdes y 2 latches para los leds rojos, sin embargo vemos que hay disponibles varios pines de los puertos a y c para activar varios circuitos latches más.

A continuación se presenta la arquitectura interna de una latch.

5.2.1. SISTEMA LÓGICO COMBINACIONAL

Un sistema lógico es aquel conjunto de elementos (unos o ceros) que describen cada una de las salidas de un sistema para todas las posibles combinaciones de entrada de los mismos.

Existen dos tipos de sistemas lógicos:

- Sistema lógico combinacional
- Sistema lógico secuencial

Los sistemas combinacionales están estructurados por un conjunto de compuertas interconectadas cuya salida está únicamente en función de la entrada, pero solo en ese mismo instante, por esta razón un sistema combinacional se dice que carece de memoria.

Un sistema combinacional puede tener n entradas y m salidas.

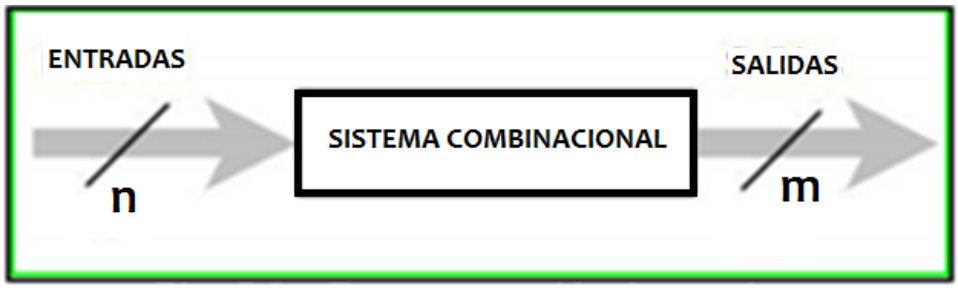


Fig. 5.2. a Representación de un sistema combinacional

Todos los circuitos combinacionales pueden representarse empleando álgebra de Boole a partir de su función lógica*.

El álgebra booleana esquematiza las operaciones lógicas Y, O , NO y SI (AND, OR, NOT, IF), así como el conjunto de operaciones unión, intersección y complemento generando de forma matemática el funcionamiento del sistema combinacional. De este modo, cada señal de entrada es una variable de la ecuación lógica de salida.

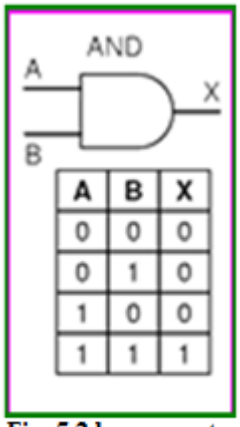
A continuación se describen cada una de las compuertas lógicas utilizadas en los sistemas combinacionales, para ejemplificar cada compuerta utilizaremos solo 2 entradas y 1 salida, cada compuerta se representa con un símbolo gráfico diferente y su operación se describe por medio de una función algebraica.

La relación entrada - salida de las variables binarias de cada compuerta se representa en una tabla de verdad.

Compuerta AND (C.I. 74LS08):

Se designan dos variables de entrada A y B y una salida binaria designada por x. La compuerta AND produce la multiplicación lógica AND. Es decir, si A y B en la entrada es 1 la salida es 1 cualquier otro caso arrogan un 0 en la salida.

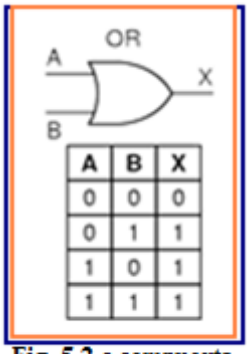
Estas condiciones se especifican en la una tabla de verdad.
El símbolo de operación algebraico de la función AND es el mismo que el símbolo de la multiplicación de la aritmética ordinaria (*).



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 5.2.b compuerta AND

Compuerta OR(C.I. 74LS32):
La compuerta OR realiza la función suma, es decir, la salida es 1 si la entrada A o la entrada B o ambas entradas son 1; de otra manera, la salida es 0.
El símbolo algebraico de la función OR (+), es igual a la operación de aritmética de suma.



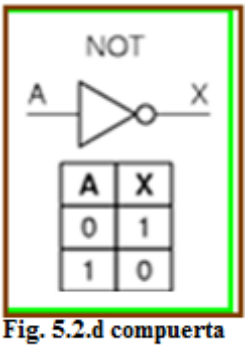
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

Fig. 5.2.c compuerta OR

Compuerta NOT(C.I. 74LS04):
El circuito NOT es un inversor que invierte el nivel lógico de una señal binaria. El símbolo algebraico utilizado para el complemento es una barra sobra el símbolo de la variable binaria.

Si la variable binaria posee un valor 0, la compuerta NOT cambia su estado al valor 1 y viceversa.

El círculo pequeño en la salida de un símbolo gráfico de un inversor designa un inversor lógico. Es decir cambia los valores binarios 1 a 0 y viceversa.



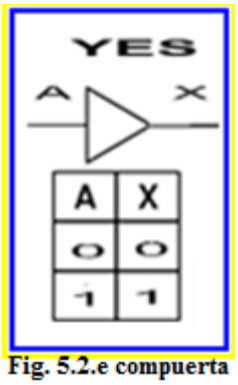
A	X
0	1
1	0

Fig. 5.2.d compuerta NOT

Compuerta Separador (yes) (C.I. 74HC4050):

Un símbolo triángulo por sí mismo designa un circuito separador, el cual no produce ninguna función lógica particular puesto que el valor binario de la salida es el mismo de la entrada.

Este circuito se utiliza simplemente para amplificación de la



A	X
0	0
1	1

Fig. 5.2.e compuerta YES

señal. Por ejemplo, un separador que utiliza 5 volt para el binario 1, producirá una salida de 5 volt cuando la entrada es 5 volt. Sin embargo, la corriente producida a la salida es muy superior a la corriente suministrada a la entrada de la misma.

De ésta manera, un separador puede excitar muchas otras compuertas que requieren una cantidad mayor de corriente que de otra manera no se encontraría en la pequeña cantidad de corriente aplicada a la entrada del separador.

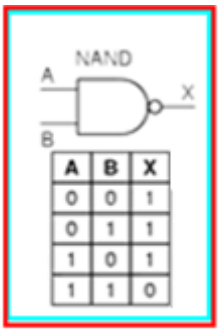


Fig. 5.2.f compuerta NAND

Compuerta NAND (C.I. 74LS00):
Esta es la compuerta complemento de la función AND gráficamente se representa por una compuerta AND seguida por un pequeño círculo.

Compuerta NOR (C.I. 74LS02):
La compuerta NOR es el complemento de la compuerta OR y utiliza el símbolo de la compuerta OR seguido de un círculo pequeño.

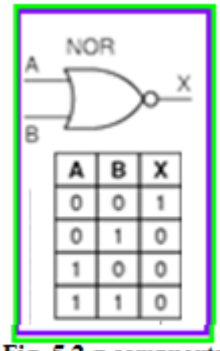


Fig. 5.2.g compuerta NOR

A continuación se presenta un par de ejemplos de sistemas combinacionales que utilizan compuertas NAND y NOR combinadas con compuertas NOT.

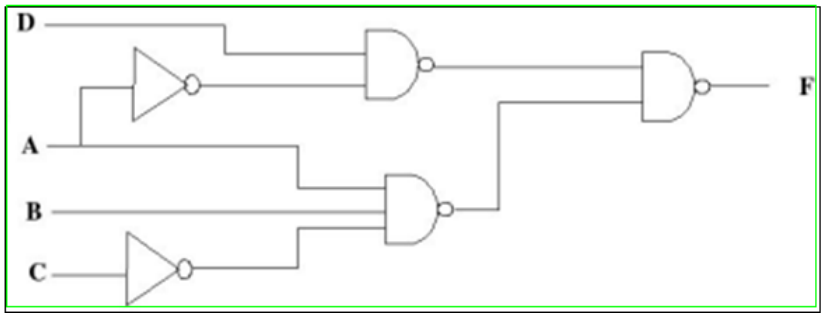
1.- Siendo la función lógica:

$$F(A,B,C,D) = \overline{A} \cdot D + B \cdot A \cdot \overline{C}$$

Que Algebraicamente se puede convertir en:

$$\begin{aligned} F(A,B,C,D) &= \overline{\overline{A} \cdot D + B \cdot A \cdot \overline{C}} \\ &= \overline{(\overline{A} \cdot D)} \cdot \overline{(B \cdot A \cdot \overline{C})} \end{aligned}$$

El diagrama del sistema quedaría de la siguiente manera:



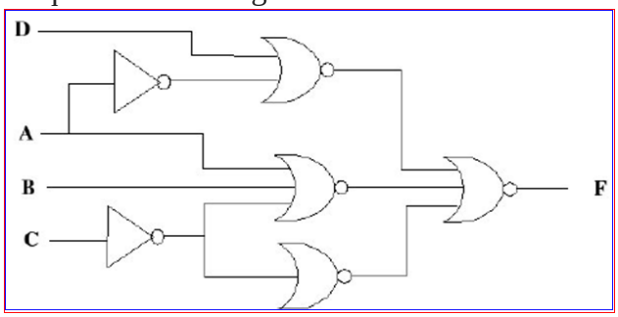
2.- Siendo la función lógica:

$$F(A,B,C,D) = (A+D)\cdot(B+A+C)\cdot C$$

Que Algebraicamente se puede convertir en:

$$\begin{aligned} F(A,B,C,D) &= \frac{(\overline{A}+D)\cdot(B+A+\overline{C})\cdot C}{\overline{(\overline{A}+D)}+\overline{(B+A+\overline{C})}+\overline{C}} \\ &= \end{aligned}$$

El diagrama del sistema quedaría de la siguiente manera:



Finalmente, tenemos que conforme se aumenten la cantidad de compuertas lógicas elementales, se construyen dispositivos lógicos altamente integrados (VLSI –Very Large Scale Integration). Estos representan varias ventajas como lo es el tamaño físico final, el costo y la latencia del circuito (retardos producidos en el acceso a los distintos componentes) debido a que las interconexiones son más rápidas, la desventaja es que se tendría que construir un chip distinto según la aplicación

5.2.2. SISTEMA LÓGICO SECUENCIAL

Partiendo de la arquitectura de los circuitos combinacionales, cuya característica principal es que las salidas solo dependen de las entradas, ahora veremos lo que es un sistema lógico secuencial.

De manera resumida, el sistema lógico secuenciales un circuito compuesto por compuertas lógicas que a su vez relacionadas entre ellas mediante líneas de retroalimentación conforman el elemento de memoria conocido como circuito biestable.

El biestable (dos estados: Q=1 y Q=0) será capaz de mantener un bit de información indefinidamente y podrá modificar su contenido en función de algún tipo de entrada.

En un sistema lógico secuencial sus salidas dependen de los valores de las entradas actuales y de los valores que hayan tomado anteriormente, desde la puesta en marcha del sistema. Es decir, de toda la secuencia de entradas almacenadas en el mecanismo de memoria desde la puesta en marcha del sistema.

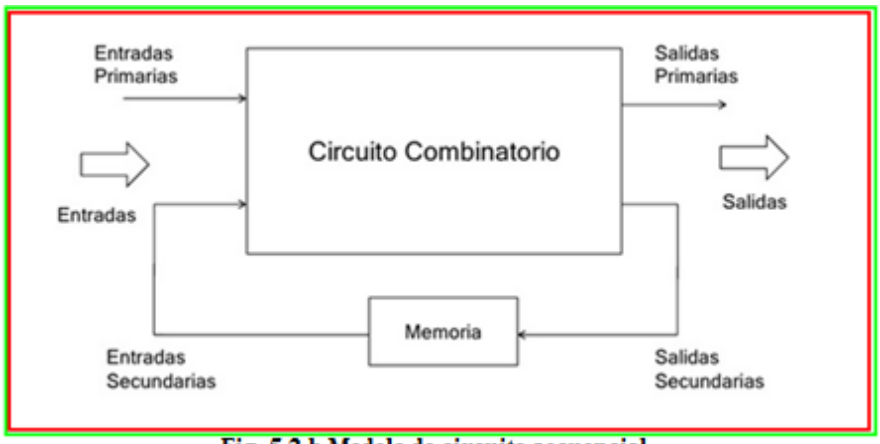


Fig. 5.2.h Modelo de circuito secuencial

5.2.2.1 BIESTABLE ELEMENTAL

El circuito más elemental capaz de mantener un valor booleano durante un tiempo indefinido es el mostrado en la siguiente figura.

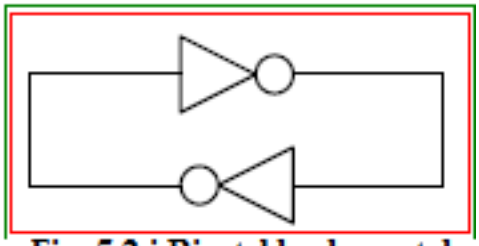


Fig. 5.2.i Biestable elemental

Si ambos inversores se orientan hacía el mismo lado, el circuito quedaría de la siguiente manera, donde Q y Q´ son sus terminales accesibles, y siempre presentarán valores complementarios.

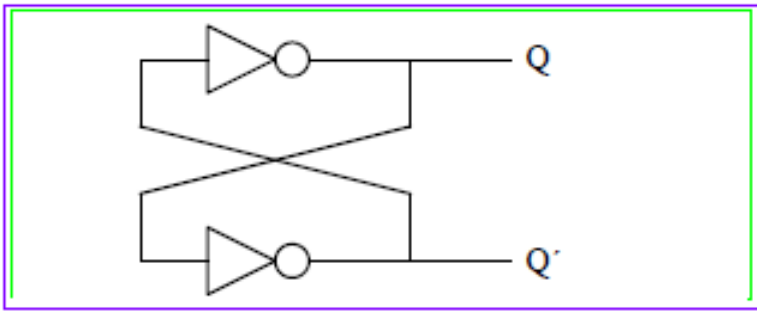


Fig. 5.2.j Biestable elemental con las terminales de salida Q y Q'

Este circuito se mantiene estable en cualquiera de los dos estados posibles de sus terminales accesibles, Cuando Q vale 1 diremos que el biestable está guardando un 1, y cuando valga 0 diremos que guarda un 0, de aquí que se llame biestable.

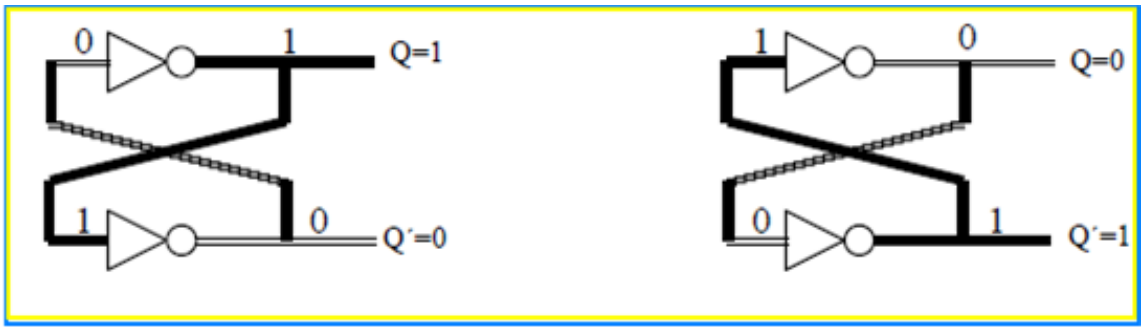


Fig. 5.2.k Biestable elemental con valores para Q =1 y Q =0, cada inversor mantiene estable la entrada del otro

Sin embargo aun cuando el circuito permite guardar un bit no permite que sea modificado.

Para poder modificar este bit, podemos hacer uso de otras compuertas lógicas y obtener diferentes tipos de circuitos biestables

5.2.2.2 BIESTABLE S-R

Si al circuito anterior cambiásemos los inversores por dos compuertas NOR tendríamos dos entradas independientes a las que denominamos S y R.

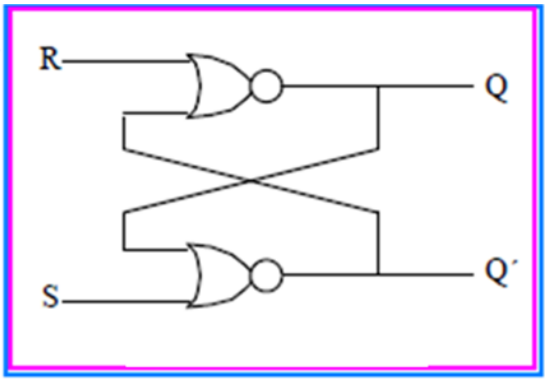


Fig. 5.2.1 Biestable S.R

Si a R y S asignamos el valor de 0, vemos que el circuito es equivalente al implementado con compuertas inversoras debido a que en la puerta NOR la operación (a+0)´= a´

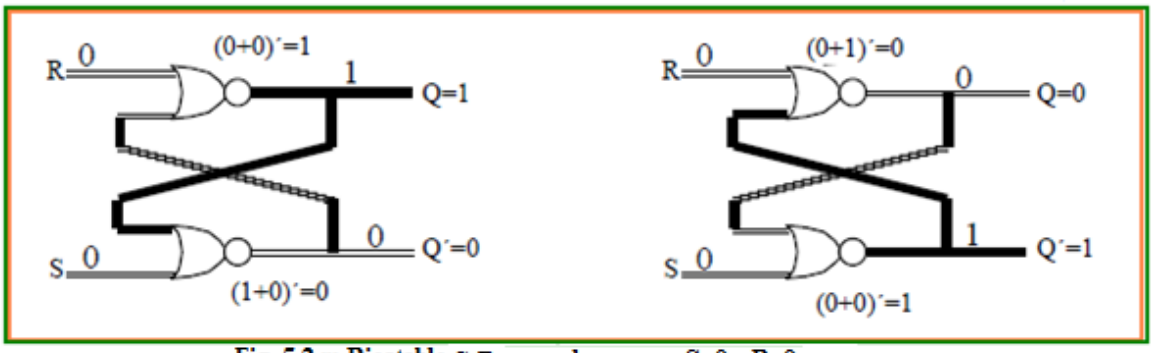


Fig. 5.2.m Biestable S-R con valores para S=0 y R=0

Con esta información podemos escribir la tabla de verdad, donde S y R son las entradas y Q^+ y Q'^+ son los valores después de pasar por las compuertas NOR,

R	S	Q	Q´	Q^+	Q'^+
0	0	0	1	0	1
0	0	1	0	1	0

Vemos que para los estados biestables de Q las salidas no fueron modificadas por tanto se dice que son estables.

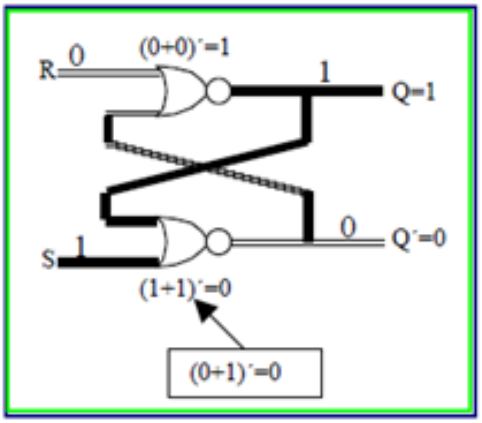


Fig. 5.2.n Biestable S-R con R =0, S=1 y Q=0 como estado inicial

Ahora veamos que sucede si R =0, S=1 y Q=0 como estado inicial:
Al activar S=1, la salida Q también se activa, de tal forma que al hacer las operaciones lógicas obtenemos los siguientes valores de Q^+ y Q'^+

Si Q=0		Q'^+	Q^+
S+Q=	1+0=1	0	
Q'^++R	0+0=0		1

Y para R =0, S=1 y Q=1 como estado inicial tenemos que:

Si Q=1		Q'^+	Q^+
S+Q=	1+1=1	0	
Q'^++R	0+0=0		1

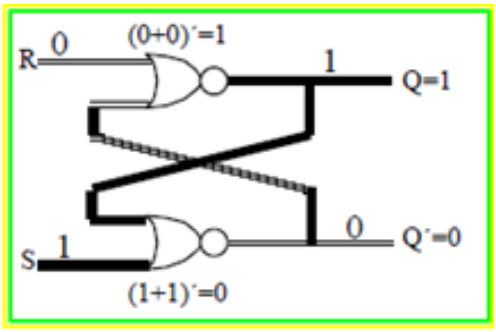


Fig. 5.2.o Biestable S-R con R =0, S=1 y Q=1 como estado inicial

La información obtenida la agregamos a la tabla de verdad del biestable R-S y quedaría de la siguiente forma:

R	S	Q	Q´	Q^+	Q'^+
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0

Si ahora invertimos los valores de R y S, es decir que R=1 y S=0, vemos quelos casos son simétricos por tanto la tabla quedaría:

R	S	Q	Q´	Q^+	Q'^+
0	0	0	1	0	1
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1

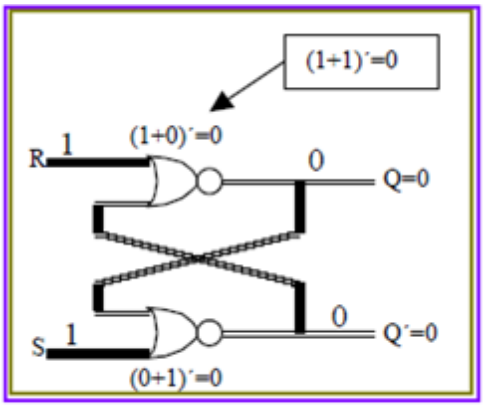


Fig. 5.2.p R=S=1 combinación prohibida

Y finalmente analizaremos el caso en que ambos valores de R=1 y S=1, este último caso no es estable por la siguiente razón:

Si Q=0		$Q^{\prime +}$	Q^{+}
S+Q=	1+0=1	0	
$Q^{\prime +}$+R	0+1=1		0

La activación de S y R nos lleva a una situación inconsistente (Q = Q´)

En consecuencia se considera la entrada R=S=1, como combinación prohibida. A través de R y S se dispone de combinaciones que permiten:

- Mantener el estado del biestable
- Modificar su valor

Por tanto, el biestable S-R cumple con la definición de biestable.
Puede servir como regla que R y S vienen de las palabras inglesas:

- Reset: reiniciar (poner a 0)
- Set: activar (poner a 1)

5.2.2.3 TABLA DE TRANSICIONES

La tabla de verdad anterior se puede condensar de la siguiente forma:

- Uniendo las líneas correspondientes a combinaciones iguales de R y de S debido a que tienen el mismo efecto sobre Q
- La información de Q´ Q se pueden deducir de Q^{+} y $Q^{\prime +}$

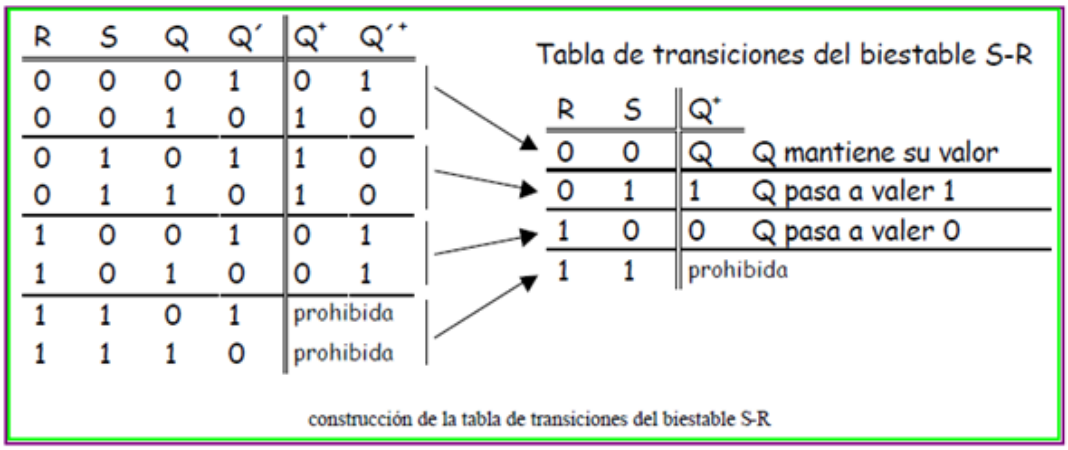
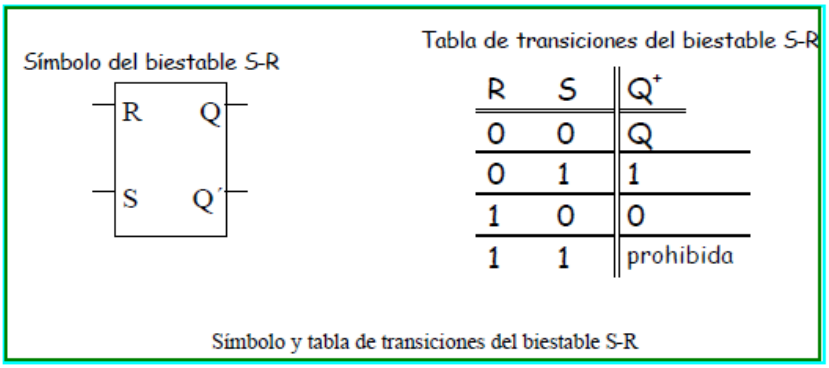


Fig. 5.2.q Tabla de transiciones

Para utilizar un biestable entonces es necesario conocer el símbolo que represente el biestable y la tabla de transiciones que defina su comportamiento. Para este flip-flop serían los siguientes:



Símbolo y tabla de transiciones del biestable S-R

Antes de continuar con el siguiente circuito biestable, es necesario conocer el concepto de sincronía.

Cuando hablamos de sincronía nos referimos a que las salidas del circuito biestableestarán en función de una señal de reloj. Esta señal del reloj puede ser interpretada de cuatro formas diferentes según el instante en el que vaya a ser considerada, véase Fig. 5.2.r.

La señal de reloj lo que hace es discretizar el tiempo, es decir, que en lugar de ver el tiempo como una dimensión continua, los circuitos lo ven como una secuencia de instantes tomando los valores 0 y 1 de manera cíclica y continua, desde el momento en que se pone en marcha el sistema hasta que se para.

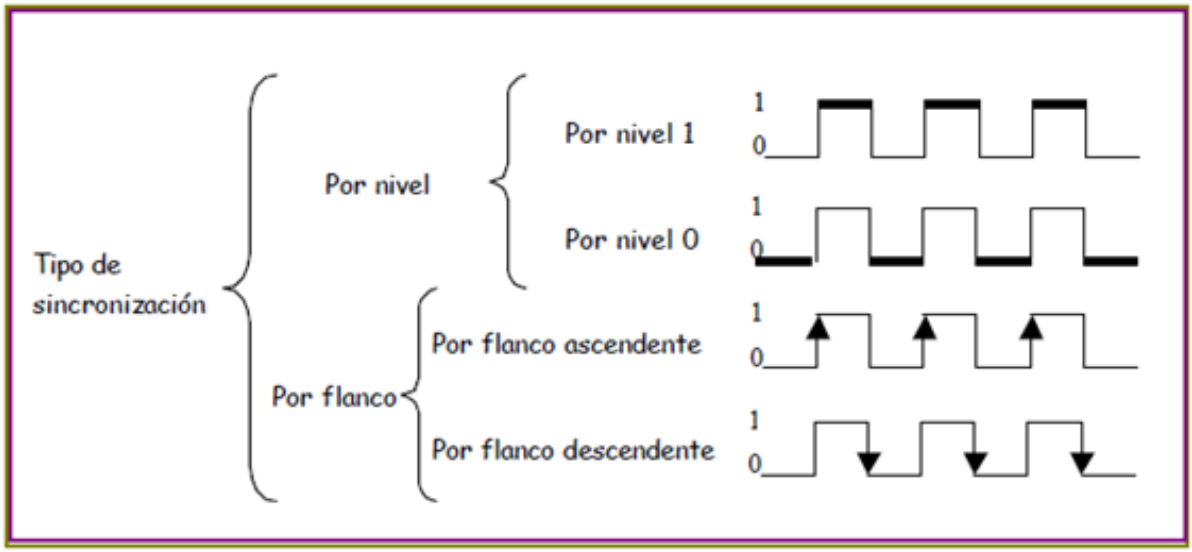


Fig. 5.2.r Tipos de sincronización

5.2.2.4 BIESTABLE S-C POR NIVEL Y POR FLANCO

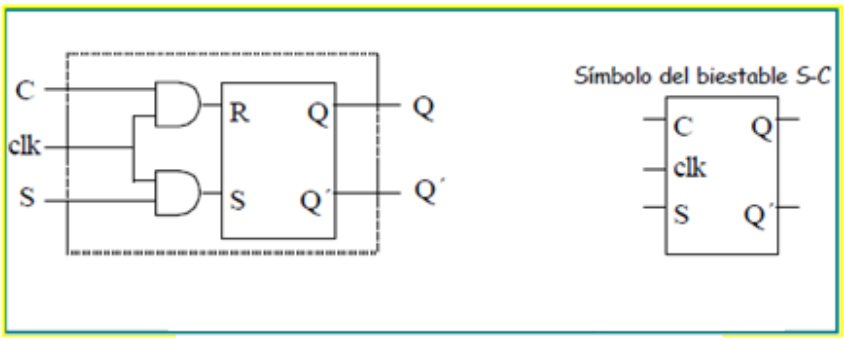


Fig. 5.2.s Biestable S-C sincronizado por nivel 1

Un biestable S-C se puede obtener a partir de un biestable S-R, este circuito es la versión sincronizada del biestable S-R y puede ser activado por nivel o por flanco:

- El biestable S-C por nivel, sólo admite cambios cuando la señal de reloj vale 1. Debido a que S y R se comunicarán directamente con S y C respectivamente.

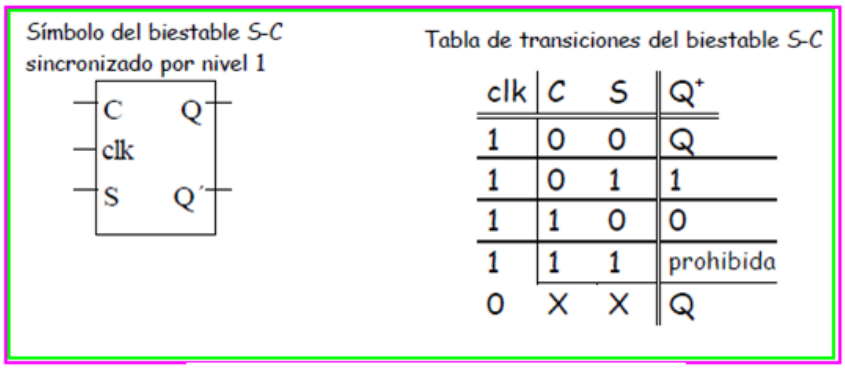


Fig. 5.2.t Símbolo y tabla de transiciones del biestable S-C

- El biestable S-C por flanco tiene otra aplicación, éste se emplea cuando para determinado tipo de circuito se necesite un biestable en el que el tiempo durante el que se pueda hacer cambios sea muy corto o lo que es lo mismo el biestable no conmute durante todo el tiempo en que el reloj está a 1 o a 0.

El hecho de que el tiempo de duración de cada pulso del reloj sea lo suficientemente corto tiene que ver con la frecuencia que pueda ser soportada. Sin embargo, el relojtiene un límite de frecuencia más allá del cual no se puede hacer que valla más rápido y que dure menos, por lo tanto la frecuencia es una limitante.

Esta limitante se omite utilizando un par de biestables sincronizados por flanco, los cuales solo conmutaran en el momento en el que el reloj pasa de 0 a 1 si es flanco ascendente ó de 1 a 0 si es flanco descendente.

Una posible forma de implementar una sincronización así es hacerlo con un biestablemaestro-esclavo. Constituido por un biestable maestro y un biestable esclavo sincronizados, uno por nivel 1, y el otro por nivel 0, con las salidas del maestro conectadas a las entradas del esclavo.

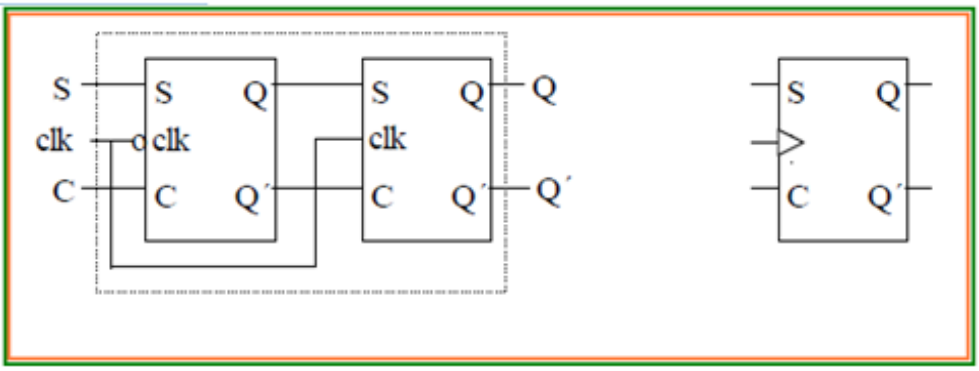


Fig. 5.2.u Biestable S-C sincronizado por flanco ascendente

La sincronización por flanco ascendente se indica con un triángulo.El nivel activo del reloj es diferente en los dos biestables por lo que no admitirán cambios al mismo tiempo.Sólo en el momento en que se active el reloj para el esclavo, éste se cargará con el último dato que tenga el maestro almacenado.

Ahora bien, tanto el biestable S-C por nivel como el biestable S-C por flanco tienen un pequeño inconveniente: ambos tienen una combinación prohibida, S=C=1.

Dadoun circuitobiestableactivado por nivel se puede construir otrobiestable a partir de él mismo, de uso más sencillo y sin combinaciones prohibidas: Este es el llamado biestable tipo D activado por nivel.

Mientras que para el circuito biestableactivado por flanco, se puede construir otrobiestable llamado J-K, este tipo de biestable añade un nivel más de retroalimentación al circuito S-C (Fig. 5.2.v) haciendo que las entradas conmuten a las salidas (estado 'toggle') y eliminando la combinación prohibida S=C=1, además acepta las condiciones no validas en donde J y K pueden ser "1" simultáneamente.

Posteriormente se puede obtener un circuito biestable tipo D activado por flanco a partir del biestable J-K aún más sencillo.

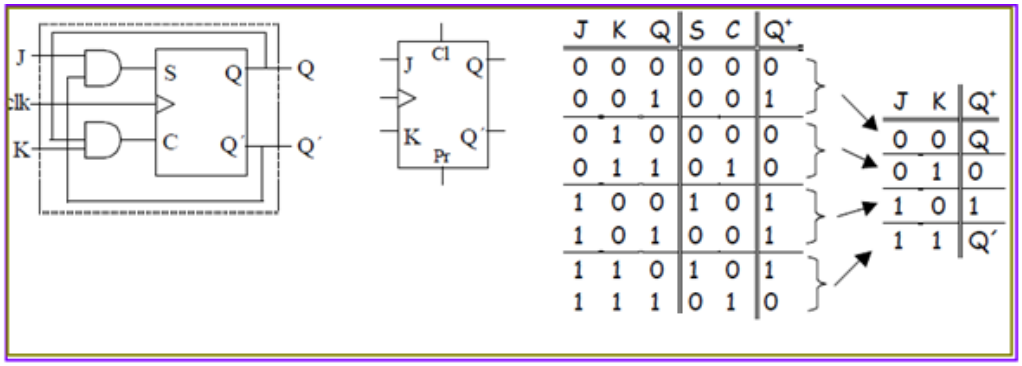


Fig. 5.2.v Biestable j-k sincronizado por flanco ascendente con entradas asincronas

5.2.2.5 BIESTABLE D POR NIVEL Y POR FLANCO.

Lo importante de los siguientes diagramas es observar que tanto en el biestable D por nivel, como en el biestable D por flanco, el valor de la salida (Q), es el mismo que el de la entrada (D) según su tabla de transiciones, esto es el elemento de memoria.

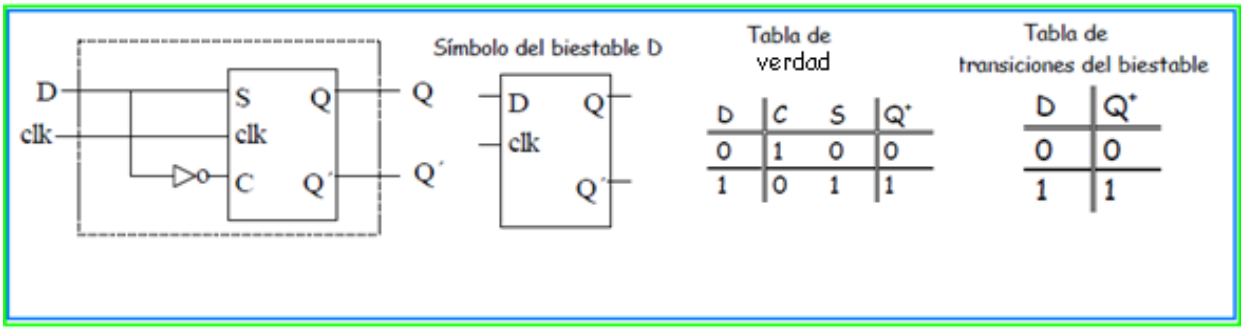


Fig. 5.2.w Biestable D sincronizado por nivel 1

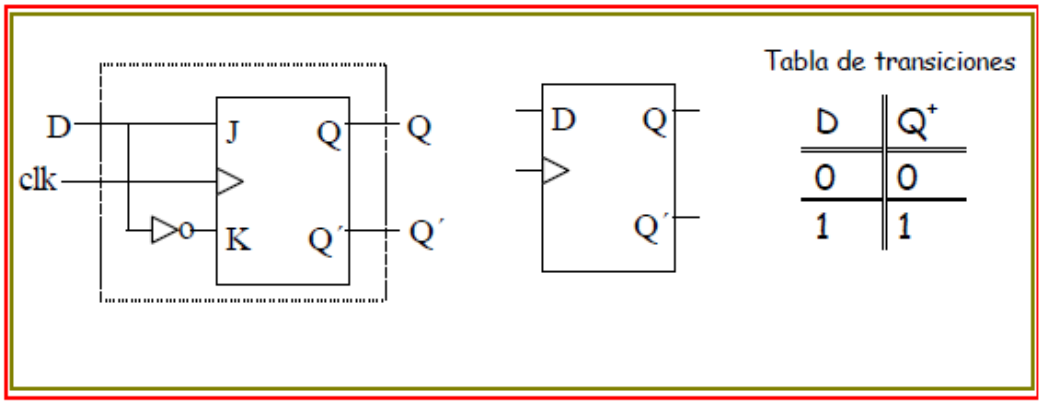


Fig. 5.2.x Biestable D sincronizado por flanco ascendente

Hasta aquí hemos visto sistemas secuenciales tipo síncronos, sin embargo existen también los sistemas digitales que operan de forma asíncrona, como veremos en el siguiente apartado.

5.2.2.6 SISTEMAS SECUENCIALES SÍNCRONOS Y ASINCRONOS.

Resumiendo, los circuitos secuenciales se clasifican de acuerdo a la manera como manejan el tiempo:

- Circuitos secuenciales síncronos
- Circuitos secuenciales asíncronos.

En el apartado anterior vimos circuitos biestables en los que se permite un cambio de estado solo en los instantes marcados por una señal de sincronismo llamada reloj, a este tipo de circuitos se les conoce como síncronos.

Pero también existen sistemas digitales que operan de forma asíncrona, en este tipo de sistemas los circuitos lógicos pueden cambiar de estado en cualquier momento en que varíen una ó más entradas, tal es el caso de los circuitos latch.

5.2.2.7 LATCH OCTAL TIPO-D 74LS573N

Una vez analizada la arquitectura de los circuitos biestables, veremos el tipo de circuito que utilizamos para el proyecto, pero antes aclarare un par de conceptos.

Existen los circuitos Latch y los circuitos Flip-flops, ambos son circuitos biestables y sirven exactamente para lo mismo, la diferencia entre ambos términos es que los latch son circuitos diseñados para trabajar con niveles (estados) y los Flip-flops para trabajar con flancos (cambio de estados)

El integrado que utilizamos es el 74LS573N, éste integrado es de tipo asíncrono lo cual significa que no necesita una señal de reloj para activar el cambio de estado del sistema, sin embargo cuenta con un pin llamado "LE" que se activa con un 1, y el cual se encarga de habilitar el latch en el momento que se requiera, y se desactiva con un 0.

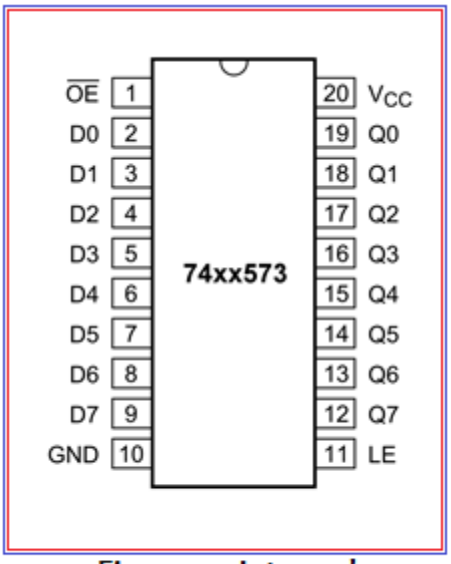


Fig. 5.2.aa Integrado 74LS573N

El latch una vez habilitado con LE en alto recibirá un byte de información de las entradas Dn, se dice que el latch es transparente porque la salida del latch cambiara en el momento en que la correspondiente entrada Dn cambia, posteriormente cuando LE esté en bajo se deshabilita el latch y se mantendrá ese byte en su puerto de salida independientemente de la entrada que le llega, solo si se le activa nuevamente volverá a tomar en cuenta el puerto de entrada y lo reflejara en el puerto de salida manteniéndolo hasta nuevo aviso.

Es importante mencionar que el pin $\overline{OE}$ debe estar en estado bajo, de esta manera el contenido de los 8 latches estará disponible en las salidas, si estuviese en estado alto las salidas del latch estarán en alta impedancia o un estado apagado. LE y $\overline{OE}$ son pines comunes a todos los latches.

FUNCTION TABLE

Inputs			Output
Output Enable	Latch Enable	D	Q
L	H	H	H
L	H	L	L
L	L	X	no change
H	X	X	Z

X = don' t care
Z = high impedance

Fig. 5.2.bb tabla de verdad 74LS573N

Los latches se pueden agruparen distintos grupos dependiendo el número de bits que puede almacenar, existen el 'latchquad' que almacena cuatro bits y el 'latch octal' para ocho bits, el 74LS573N es de este tipo, esta característica nos permite enumerar ocho camas con un solo latch.

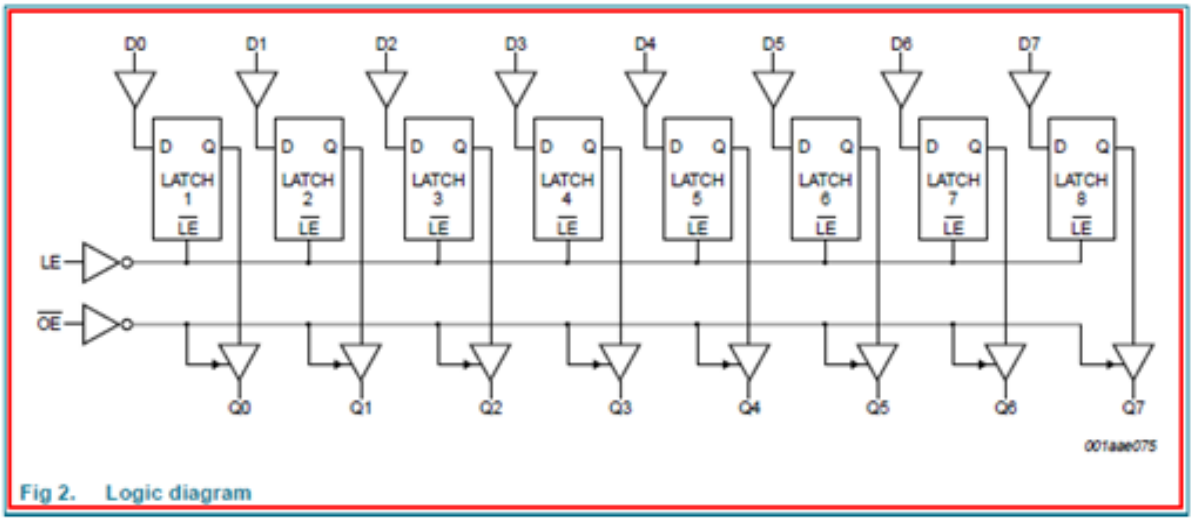


Fig. 5.2.cc Diagrama lógico latch octal 74LS573N

5.3 MEMORIA EEPROM INTERNA DEL PIC

5.3.1 INTRODUCCIÓN

Una vez simulado en PROTEUS lo que hasta el momento tenemos, surgió un nuevo problema. A continuación se ejemplifica y se analiza la cuestión.

Suponiendo que el valor de mi cama es 1, este valor es procesado por el microcontrolador y enviado al latch.

Primero habilito el latch que tiene el número de cama 1 asignado, una vez habilitado enciendo el pin correspondiente del microcontrolador para encender el pin D0 del latch, el led que indica la cama 1 prende, seguidamente deshabilito el latch y apago el pin del microcontrolador que mando un estado alto al pin D0.

Esto último, tiene la finalidad de que si se activa un latch diferente, el pin del micro que activa a D0 no encienda a D0 de otro latch, representando un número de cama indeseado.

Continuando con el ejemplo, hasta el momento el pin de habilitación del latch y todas las entradas están a 0, nuevamente, si realizo cualquier otra llamada (cama 2 - cama 8) sucederá lo siguiente:

Originalmente el led que indica la cama 1 esta encendido porque fue el valor 1 que almaceno el latch, al activar el latch nuevamente y tener en todas las entradas el valor de cero a excepción del nuevo número de cama, se prendera el led que indica el nuevo número de cama pero se apagara el led de la cama 1. Este ejemplo es aplicable para cualquier número de cama y cualquier combinación. Es por ello que recurrí a la memoria EEPROM interna del PIC.

5.3.2 ESTRUCTURA INTERNA DEL MICROCONTROLADOR

Para comprender el funcionamiento de la EEPROM, se repasan los tipos de que memoria que existen. Hay dos tipos de arquitecturas conocidas; la de von Neumann, y la arquitectura Harvard.

- Arquitectura Von Neumann: Dispone de una sola memoria principal donde se almacenan datos e instrucciones de forma indistinta. A dicha memoria se accede a través de un sistema de buses único (direcciones, datos y control).

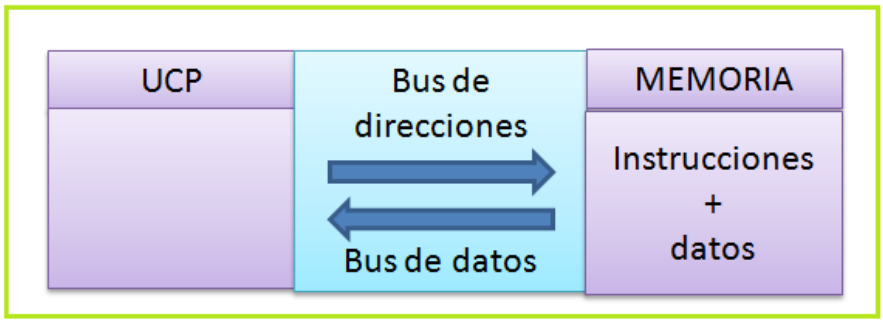


Fig. 5.3.a Arquitectura Von Neumman

- Arquitectura Harvard: Dispone de dos memorias independientes, una que contiene sólo instrucciones, y otra que contiene sólo datos. Ambas disponen de sus respectivos sistemas de buses de acceso y es posible realizar operaciones de acceso (lectura o escritura) simultáneamente en ambas memorias, ésta es la estructura para los PICs.

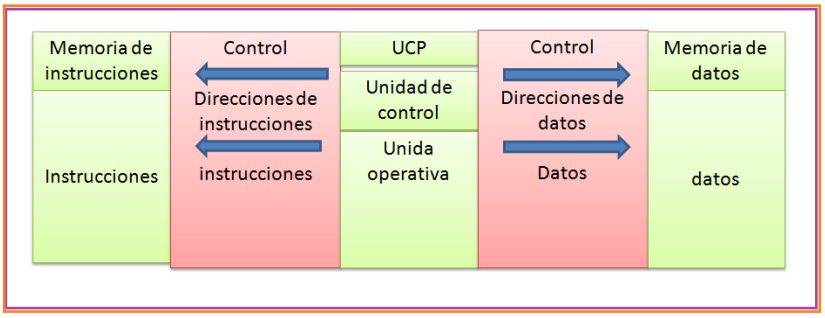


Fig.5.3.b. Arquitectura Harvard

- Procesador o UCP

Es el elemento más importante del microcontrolador. Se encarga de direccionar la memoria de instrucciones, recibir el código de la instrucción en curso, decodificarlo y ejecutarlo, también realiza la búsqueda de los operandos y almacena el resultado.

5.3.3 MEMORIA DE PROGRAMA

Esta memoria sería la memoria de instrucciones, también puede ser llamada ROM (read only memory) aquí se almacena el programa o código que el micro debe ejecutar. Es del tipo no volátil, es decir el programa se mantiene aunque desaparezca su alimentación. No hay posibilidad de utilizar memorias externas de ampliación, existen dos tipos de memoria de programa:

Memoria EEPROM de instrucciones. (Electrical Erasable Programmable Read Only Memory / Memoria de sólo lectura Programable y borrable eléctricamente). Esto se hace a través de un circuito grabador y bajo el control de un PC. El número de veces que puede grabarse y borrarse una memoria EEPROM es finito aproximadamente 1000 veces. Este tipo de memoria es relativamente lenta y ya no existe en el PIC16f886 que utilizamos.

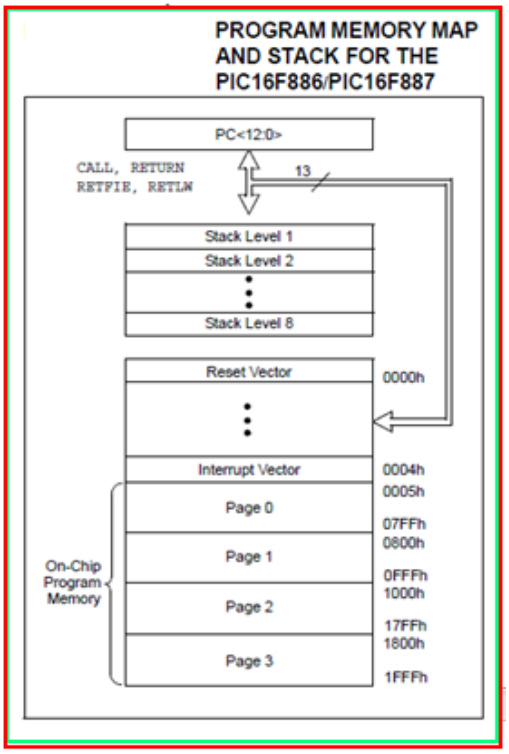


Fig. 5.3.c Mapa de memoria de programa del PIC16f886

- Memoria FLASH.

Un tipo de memoria ROM más moderna es la memoria FLASH, ésta se puede borrar y volver a programar varias veces, aun siendo descrita como "memoria de sólo lectura". En el PIC16f886A que utilizamos ya se encuentra disponible y tiene las mismas características que la EEPROM, pero ésta tiene menor consumo de energía y mayor capacidad de almacenamiento, por ello está sustituyendo a la memoria EEPROM.

La memoria del programa o Flash se divide en paginas según el siguiente diagrama, en ellas se almacena el código del programa.

El Stack es un grupo de localidades de lectura – escritura que son usadas para almacenar el contenido del contador de programa (registros y direcciones de memoria) temporalmente durante la ejecución del programa, así cuando se efectúa una llamada (CALL) o una interrupción, el contador de programa (CP) sabe donde debe regresar para continuar con la ejecución del programa. El Stack es como una pila de 8 platos donde el último en poner es el primero en sacar.

5.3.4 CONTADOR DE PROGRAMA

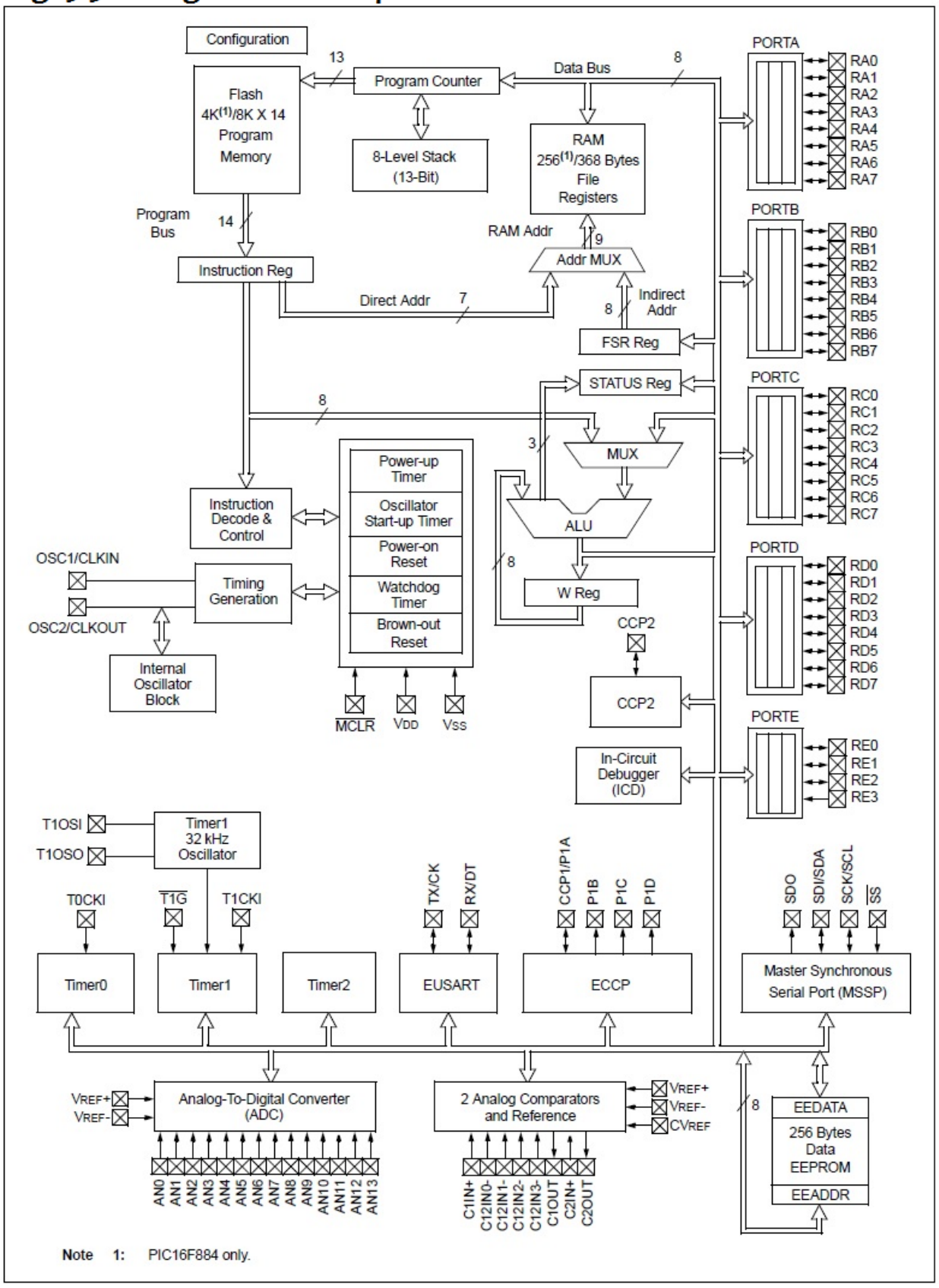
Es un registro interno que se utiliza para direccionar las instrucciones del programa de control que están almacenadas en la memoria del programa. Este registro contiene la dirección dela próxima instrucción a ejecutar y se incrementara automáticamente de manera que la secuencia natural de ejecución del programa es lineal, una instrucción después de la otra.

El PIC tiene un contador de programa de 13 bits (2^{13}=8192 direcciones) capaz de direccionar una memoria de programa de 8K (8x1024=8192 instrucciones) con un ancho de 14 bits cada una.

Cuando ocurre un Reset, el contador de programa (PC) apunta a la dirección 0000h, y el micro se inicia nuevamente. Por esta razón, en la primera dirección del programa se debe escribir todo lo relacionado con la iniciación del mismo.

Ahora, si ocurre una interrupción el contador de programa (PC) apunta a la dirección 0004h, entonces ahí escribiremos la programación necesaria para atender dicha interrupción.

Fig. 5.3.d Diagrama de bloques del PIC16f886



Note 1: PIC16F884 only.

5.3.5 MEMORIA DE DATOS

La memoria de datos se divide en dos, la SRAM y la memoria EEPROM. Básicamente la diferencia radica en que la SRAM es del tipo volátil y la EEPROM guarda los datos de forma permanente,esdeciraun cuando no haya una fuente de alimentación.

PIC16F882/883/884/886/887

Device	Program Memory	Data Memory		I/O	10-bit A/D (ch)	ECCP/ CCP	EUSART	MSSP	Comparators	Timers 8/16-bit
	Flash (words)	SRAM (bytes)	EEPROM (bytes)							
PIC16F882	2048	128	128	28	11	1/1	1	1	2	2/1
PIC16F883	4096	256	256	24	11	1/1	1	1	2	2/1
PIC16F884	4096	256	256	35	14	1/1	1	1	2	2/1
PIC16F886	8192	368	256	24	11	1/1	1	1	2	2/1
PIC16F887	8192	368	256	35	14	1/1	1	1	2	2/1

Fig. 5.3.e Tamaño de la memoria de datos del PIC16f886

5.3.5.1 MEMORIA DE DATOS SRAM

La memoria de datos SRAM o RAM estática se destina a guardar las variables y datos. Es volátil, es decir los datos almacenados se borran cuando desaparece la alimentación. Los datos almacenados en ésta memoria varían continuamente, por lo que esta memoria debe ser de lectura y escritura.

La RAM del PIC16f886 esta particionada en múltiples bancos los cuales pueden extenderse hasta128 bits cada uno (7Fh) y estos contienen:

- los Registros de funciones especiales (SFR). Son los primeros registros, cada uno cumple con un propósito especial en el control del microcontrolador.

- y los Registros de Propósito General (GPR). Son registros de uso general que se pueden usar para guardar los datos temporales del programa que se esté ejecutando.

Los bits RP1 RP0 son los bits para seleccionar el banco.

Para seleccionar el banco a acceder hay que configurar el bit 5 (RP0) y el bit 6 (RP1) del registro STATUS según la siguiente tabla.

RP1:RP0	Bank
00	0
01	1
10	2
11	3

Fig. 5.3.f Configuracion de RP1:RP2 para la seleccion del banco a utilizar

REGISTER 2-1: STATUS: STATUS REGISTER

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC(1)	C(1)
bit 7							bit 0

Fig. 5.3.g Registro STATUS

Las primeras direcciones de cada banco están reservadas para los Registro de Funciones Especiales. En las direcciones posteriores a la de los Registros de funciones especiales están los registros de uso general.

Todos los bancos implementados contienen Registros de funciones especiales. Algunos de los Registros de funciones especiales de un banco se encuentran duplicados en la misma dirección de otro banco para la reducción de código y un acceso más rápido.

Fig. 5.3.h Registro de funciones especiales del PIC16f886

	File Address		File Address		File Address		File Address
Indirect addr. (1)	00h	Indirect addr. (1)	80h	Indirect addr. (1)	100h	Indirect addr. (1)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h	WDTCON	105h	SRCON	185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	CM1CON0	107h	BAUDCTL	187h
PORTD(2)	08h	TRISD(2)	88h	CM2CON0	108h	ANSEL	188h
PORTE	09h	TRISE	89h	CM2CON1	109h	ANSELH	189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDAT	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2(1)	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved	18Eh
TMR1H	0Fh	OSCCON	8Fh	EEADRH	10Fh	Reserved	18Fh
T1CON	10h	OSCTUNE	90h		110h		190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h	WPUB	95h		115h		195h
CCPR1H	16h	IOCB	96h	General Purpose Registers	116h	General Purpose Registers	196h
CCP1CON	17h	VRCON	97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h	16 Bytes	119h	16 Bytes	199h
RCREG	1Ah	SPBRGH	9Ah		11Ah		19Ah
CCPR2L	1Bh	PWM1CON	9Bh		11Bh		19Bh
CCPR2H	1Ch	ECCPAS	9Ch		11Ch		19Ch
CCP2CON	1Dh	PSTRCON	9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
	20h		A0h		120h		1A0h
General Purpose Registers	3Fh	General Purpose Registers		General Purpose Registers		General Purpose Registers	
	40h						
96 Bytes		80 Bytes		80 Bytes		80 Bytes	
	6Fh		EFh		16Fh		1EFh
	70h	accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h	accesses 70h-7Fh	1F0h
	7Fh		FFh		17Fh		1FFh
Bank 0		Bank 1		Bank 2		Bank 3	

Unimplemented data memory locations, read as '0'.

Note 1: Not a physical register.

2: PIC16F887 only.

5.3.5.2 EEPROM

Esta memoria sirve para guardar la información que se genera durante el proceso de forma permanente, es decir, los datos permanecerán incluso cuando el sistema se desconecta de la alimentación. El PIC16f886 dispone de un área de datos EEPROM no volátil. Se pueden realizar en ella dos tipos de operaciones:

- operación de lectura
- operación de escritura o grabación.

Un ciclo de grabación en una posición EEPROM de datos dura unos 10 ms, un tiempo muy elevado para la velocidad del procesador, que se controla mediante un temporizador interno, Al escribir en una posición de memoria ya ocupada, automáticamente se borra el contenido que había y se introduce el nuevo dato, por lo que no hay comando de borrado. Según la hoja de características del PIC, éste puede soportar hasta un millón de ciclos de escritura/borrado de su memoria EEPROM de datos y es capaz de guardar la información inalterada durante más de 40 años

Esta memoria no está directamente en el espacio del registro de archivos, ésta se direcciona indirectamente a través de los registros de funciones especiales. Hay seis SFR utilizados para leer y escribir en esta memoria:

• EECON1 (EEPROM CONTROL REGISTER 1). Los bits de este registro definen el modo de funcionamiento de la memoria. Los bits RD y WR del EECON1 indican respectivamente lectura y escritura. No hay que ponerlos a “0” sólo a “1”, ya que se borran automáticamente cuando la operación de lectura o escritura ha sido completada.
• EECON2 (EEPROM CONTROL REGISTER 2). Éste registro no está implementado físicamente, por lo que es imposible leerlo (si se intenta leer todos sus bits se leen como ceros). Se emplea como dispositivo de seguridad durante el proceso de escritura de la EEPROM, para evitar las interferencias en el largo intervalo de tiempo que precisa su desarrollo.
• EEDATA (EEPROM DATA REGISTER). Contiene los bytes que se van a escribir o que se han leído de la EEPROM de datos.
• EEADR (EEPROM ADDRES REGISTER). Contiene la dirección de la EEPROM de datos a la que acceder para leer o escribir. Las 256 posiciones (para el PIC16f886) de memoria EEPROM ocupan las direcciones de un mapa que comienza en la posición 00h hasta la FFh.
• EEDATH
• EEADRH (únicamente 4 bits para el PIC16F886/PIC16F887)

Estos dos últimos registros trabajan en conjunto con EEDATA y EEADR respectivamente en el PIC16F887 y el PIC16F886. Los registros EEDAT y EEDATH forman una palabra de dos bytes que contiene los datos de 14 bits para lectura / escritura y los registros EEADR y EEADRH forman una palabra de 2 bytes que contiene la dirección (de doce bits)de la ubicación que está siendo leída de la EEPROM

Fig. 5.3.i Resumen del Banco 2 del registro de funciones especiales del PIC16f886

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 2											
100h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	37,213
101h	TMR0	Timer0 Module Register								xxxx xxxx	73,213
102h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	37,213
103h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	29,213
104h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	37,213
105h	WDTCON	—	—	—	WDTPS3	WDTPS2	WDTPS1	WDTPS0	SWDTEN	---0 1000	221,214
106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	48,213
107h	CM1CON0	C1ON	C1OUT	C1OE	C1POL	—	C1R	C1CH1	C1CH0	0000 -000	88,214
108h	CM2CON0	C2ON	C2OUT	C2OE	C2POL	—	C2R	C2CH1	C2CH0	0000 -000	89,214
109h	CM2CON1	MC1OUT	MC2OUT	C1RSEL	C2RSEL	—	—	T1GSS	C2SYNC	0000 --10	91,215
10Ah	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	37,213
10Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF(1)	0000 000x	31,213
10Ch	EEDAT	EEDAT7	EEDAT6	EEDAT5	EEDAT4	EEDAT3	EEDAT2	EEDAT1	EEDAT0	0000 0000	112,215
10Dh	EEADR	EEADR7	EEADR6	EEADR5	EEADR4	EEADR3	EEADR2	EEADR1	EEADR0	0000 0000	112,215
10Eh	EEDATH	—	—	EEDATH5	EEDATH4	EEDATH3	EEDATH2	EEDATH1	EEDATH0	--00 0000	112,215
10Fh	EEADRH	—	—	—	EEADRH4(2)	EEADRH3	EEADRH2	EEADRH1	EEADRH0	---- 0000	112,215

Legend: – = Unimplemented locations read as '0', u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented
Note 1: $\overline{MCLR}$ and WDT Reset does not affect the previous value data latch. The RBIF bit will be cleared upon Reset but will set again if the mismatch exists.
2: PIC16F886/PIC16F887 only.

Fig. 5.3.ii Resumen del Banco 3 del registro de funciones especiales del PIC16f886

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 3											
180h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	37,213
181h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	30,214
182h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	37,213
183h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	29,213
184h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	37,213
185h	SRCON	SR1	SR0	C1SEN	C2REN	PULSS	PULSR	—	FVREN	0000 00-0	93,215
186h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	48,214
187h	BAUDCTL	ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN	01-0 0-00	160,215
188h	ANSEL	ANS7(2)	ANS6(2)	ANS5(2)	ANS4	ANS3	ANS2	ANS1	ANS0	1111 1111	40,215
189h	ANSELH	—	—	ANS13	ANS12	ANS11	ANS10	ANS9	ANS8	--11 1111	99,215
18Ah	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	37,213
18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF(1)	0000 000x	31,213
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	113,215
18Dh	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	111,215

Legend: – = Unimplemented locations read as '0', u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented
Note 1: $\overline{MCLR}$ and WDT Reset does not affect the previous value data latch. The RBIF bit will be cleared upon Reset but will set again if the mismatch exists.
2: PIC16F884/PIC16F887 only.

El PIC16f886que utilizamos tiene en particular 256 bytes de EEPROM de datos con un rango de direcciones de 00h a FFh. En los dispositivos con 128 bytes, el rango de direcciones va de 80h a FFh. Al escribir en lugares no permitidos un circuito de chargepump* en el chip se apagara.

Las tensiones escritura / borrado son generadas por un chargepump* dentro del chip estimado para que funcione en el rango de tensiones del dispositivo. Cuando existe un código de protección del dispositivo, la CPU puede continuar leyendo y escribiendo en la memoria EEPROM de datos.

Dependiendo de la configuración de los bits de protección contra escritura, el dispositivo puede o no puede ser capaz de escribir ciertos bloques de la memoria de programa; Sin embargo, está permitido leer la memoria. Cuando existe un código de protección, el programador del dispositivo no puede acceder a la memoria de datos o la memoria de programa; sin embargo esto no inhibe la lectura o escritura interna.

5.3.6 PIC-C EN LA PROGRAMACIÓN DE LA EEPROM INTERNA DEL PIC16F886A

Como se vio en el sub-capitulo anterior,la memoria EEPROMesdireccionada indirectamente a través de los registros de funciones especiales (SFR), si esto se programara en ensamblador se tendrían que llevar a cabo una serie de pasos para leer y escribir en la EEPROM. Sin embargo CCS nos permite aplicar solo un par de funciones para leer y escribir en donde solo se especifican los parámetros necesarios de dichas funciones y son las siguientes:

- value = read_EEPROM (address): función básica para leer el valor de la EEPROM interna del PIC. Devuelve un valor entero (int8) de un byte. "address" puede ser un entero de 8 ó 16 bit. Dependiendo del PIC que utilicemos dispondremos de másó menos memoria EEPROM, por ejemplo el PIC 16f84A dispone de 64 bytes y el PIC16F886 tiene 256 bytes que se direccionan del 0 a 255.

- write_EEPROM (address, value): esta función escribe un dato (entero de 8 bit) en la dirección especificada en address en la memoria interna del PIC. Al igual que en la instrucción read_EEPROM “address” puede ser un entero de 8 ó 16 bit.

Algunos dispositivos permiten leer y escribir datos en la memoria de programa en tiempo de ejecución, para los dispositivos que soportan esta funcionalidad CCS, nos proporciona las siguientes funciones:

- value = read_program_EEPROM (address): esta función lee un dato de la memoria de programa del PIC y devuelve el valor leído como un entero de 16

bits. Address es un entero de 16 ó 32 bits que depende del dispositivo empleado.

- write_program_EEPROM (address, data): función homologa a la anterior pero que nos permite escribir datos en la memoria de programa. data tiene que ser un entero de 16 bits.

Con este par de funciones se desarrollo el siguiente código, que nos permite solucionar el problema planteado al inicio del capítulo.

5.3.7 CÓDIGO FUENTE PARA EL CONTROL DE LOS LATCH.

Recordemos, el PIC esclavo va a recibir un cadena de datos del PIC maestro, y almacenara en las variables "latchin" y "call" el número de cama y el tipo de llamada respectivamente, ambas variables las enviara ya sea a la subrutina "loop1()" ó a la subrutina "loop2()" ; esto dependerá del número de cama, ya que de la cama 1 a la 8 los datos los envía a "loop1()" y de la cama 9 a la 16 los datos se envían a la subrutina "loop2()".

```
26  }
27          if (dato[1]=='1'){
28             latchin=dato[2];
29             call=dato[1];
30             loop1();
31             loop2();
32             }
33          if (dato[1]=='4'){
34             latchin=dato[2];
35             call=dato[1];
36             loop1();
37             loop2();
38             }
39          if (dato[1]=='*'){
40             latchin=dato[2];
41             call=dato[1];
42             loop1();
43             loop2();
44             }
```

Una vez enviados los datos, se procederá de la siguiente manera para trabajar con ellos, El siguiente fragmento de código ilustra solo un ejemplo, para este caso solo utilizaremos la cama 6, sin embargo se puede extrapolar su programación para cada una de las camas.

latchmem3.c

```
#byte puerto_b=0x06
int8 latchin, call;
void rutina1();
void rutina2();
void rutina3();
void rutina4();
void portboff();
/************************************ ************************************/
void loop1 (void){
port_b_pullups(false);
switch (latchin){
//////////////////////////////// CASO 6 ////////////////////////////////////////
case 6:
   if (call =='1'){
       write_eeprom (5,1);
       if (read_eeprom(5)==1){
                          output_high(PIN_c0);
                          if (read_eeprom(0)== 1){
                          output_high(PIN_b0);}      //1 VERDE ON //NEW
                          IF (read_eeprom(1)== 1){   //2 VERDE ON //NEW
                          output_high(PIN_b1);}
                          IF (read_eeprom(2)== 1){   //3 VERDE ON
                          output_high(PIN_b2);}
                          IF (read_eeprom(3)== 1){   //4 VERDE ON
                          output_high(PIN_b3);}
                          if (read_eeprom(4)== 1){
                          output_high(PIN_b4);}      //5 VERDE ON //NEW
                          if (read_eeprom(5)== 1){
                          output_high(PIN_b5);}      //6 VERDE ON //NEW
                          IF (read_eeprom(6)==1){    //7 VERDE ON
                          output_high(PIN_b6);}
                          if (read_eeprom(7)== 1){
                          output_high(PIN_b7);}      //8 VERDE ON //NEW
                          Output_low(PIN_c0);
                                 output_low(PIN_b0);
                                 output_low(PIN_b1);
                                 output_low(PIN_b2);
                                 output_low(PIN_b3);
                                 output_low(PIN_b4);
                                 output_low(PIN_b5);
                                 output_low(PIN_b6);
                                 output_low(PIN_b7);
                          output_low(PIN_b7);
                        output_high(PIN_a0);
                          if (read_eeprom(0)==2) {
                          output_high(PIN_b0);}     //1 ROJO ON //NEW
                          if (read_eeprom(1)==2) {
                          output_high(PIN_b1);}     //2 ROJO ON //NEW
                          if (read_eeprom(2)==2){   //3 ROJO ON
                          output_high(PIN_b2);}
                          if (read_eeprom(3)==2){   //4 ROJO ON
                          output_high(PIN_b3);}
                          if (read_eeprom(4)==2){   //5 ROJO ON
                          output_high(PIN_b4);}
                          if (read_eeprom(5)==2){   //6 ROJO ON
                          output_high(PIN_b5);}
                          if (read_eeprom(6)==2){   //7 ROJO ON
                          output_high(PIN_b6);}
                          if (read_eeprom(7)==2){   //8 ROJO ON
                          output_high(PIN_b7);}
                          output_low(PIN_a0);
                                  portboff();
```

rutina2()

portboff()

rutina1()

```
62  }
63  }
64      if (call =='4'){
65          write_eeprom (5,2);
66          if (read_eeprom(5)==2){
67                              output_high(PIN_A0);
68                                rutina1();                  //1-8 ROJO ON
69                                output_high(PIN_c0);
70                                rutina2();
71
72  }
73  }
74      if (call =='*'){
75
76              write_eeprom (5, 3);
77                              output_high(PIN_c0);
78                              rutina2();
79                                output_high(PIN_a0);
80                                rutina1();                  //1-8 ROJO ON
81      }
82            break;
83  } //FIN DE SWITCH
84  } //FIN DE LOOP();
```

COMENTARIO DEL CODIGO:

Primero que nada, cree un archivo que contendrá las rutinas para operar los datos “latchin” y “call”, al que he llamado “latchmem3.c”, este a su vez contendrá otras subrutinas que llamadas “rutina1()”,”rutina2()”,”rutina3()”,”rutina4()”, de las cuales para este ejemplo solo utilizaremos “rutina1()” y ”rutina2()” y una subrutina adicional llamada “portoff()”.

Al tratarse de la cama 6 por lo tanto se utiliza la rutina principal de loop1() y no la de loop2(), La sentencia que utilizo es la SWITCH y se describe a continuación:

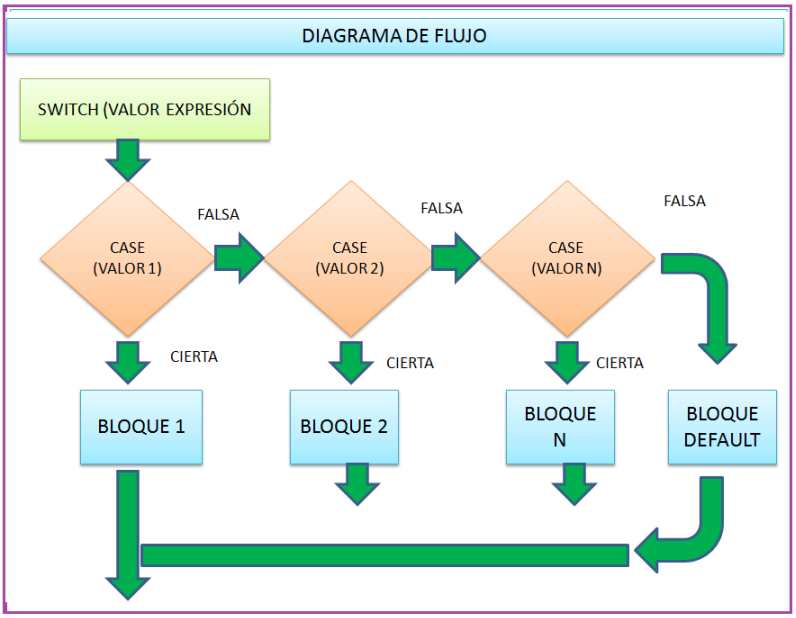


Fig. 5.3.j Sentencia Switch

Continuando con el código, tenemos que se trata del Case 6, que representa a la cama 6 y dependiendo el tipo de llamada la variable “call” tendra tresopciones:

- el caso en el que la variable call ==´1´ y que indica una llamada normal
- El caso donde la variable call==´4´ y que indica la llamada de urgencia y
- Finalmente cuando la variable call==´*´que deshabilita ambas llamadas.

Ejemplificare el caso en que se trata de un una llamada normal:

```
case 6:
 if (call =='1')
```

la instrucción write_EEPROM (5,1) indicara que se escribira un valor de 1 en la dirección 5 de la EEPROM, posteriormente se leera la dirección 5 y se comprobara si existe un dato en esa direccion, como dicha condición siempre se cumple entonces se continua con el programa.

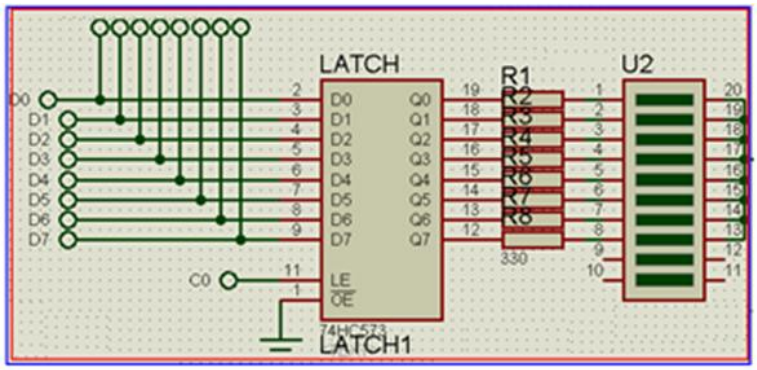


Fig. 5.3.k Pin de activación del latch a utilizar

El programa inicia encendiendo el pin C0 (este pin si recordamos es el que habilita el latch 1 donde se encuetra el led que representa la cama 6) para despues preguntar desde la dirección de memoria 0 hasta la 7 si se tiene una valor de 1 en cualquiera de estas direcciones, en caso de que la condición del IF se cumpla, el microcontrolador manda un alto al pin o pines correspondientes del puerto B que representen las camas activadas, acto seguido se deshabilita ese latch y se madan a bajo todos los pines del puerto B, con la finalidad de que al habilitarse un latch diferente, no se encienda ningun led del nuevo latch.

Hasta este momento solo se ha tomado en cuenta las llamadas normales, pero ¿que sucedería si se tuviesen combinadas llamadas normales y de urgencia? La siguiente parte del código responde esta pregunta.

Se enciende el pin A0, éste pin habilita el latch que enciende los leds rojos (cama 1 a la 8) y nuevamente el código pregunta desde la dirección 0 hasta la 7 de la EEPROM si se cumple que en alguna de ellas se tenga un valor de 0 (el valor de 0, al igual que para el ejemplo de los leds verdes en donde se tiene un valor de 1 fueron asignado por mi para identificar si enciende un led verde o uno led rojo), en caso de cumplirse esta condición se activa el pin correspondiente.

De esta manera, el led de la cama 6 queda encendido , junto con cualquier otro numero de cama ya sean de los leds verdes o rojos.

Nuevamente, se apaga el pin A0 para deshabilitar el latch y se apagan todos los pines del puerto B del microcontrolador.

```
                output_low(PIN_b7);
              output_high(PIN_a0);
rutina1()     if (read_eeprom(0)==2) {
              output_high(PIN_b0);}     //1 ROJO ON //NEW
              if (read_eeprom(1)==2) {
              output_high(PIN_b1);}     //2 ROJO ON //NEW
              if (read_eeprom(2)==2){   //3 ROJO ON
              output_high(PIN_b2);}
              if (read_eeprom(3)==2){   //4 ROJO ON
              output_high(PIN_b3);}
              if (read_eeprom(4)==2){   //5 ROJO ON
              output_high(PIN_b4);}
              if (read_eeprom(5)==2){   //6 ROJO ON
              output_high(PIN_b5);}
              if (read_eeprom(6)==2){   //7 ROJO ON
              output_high(PIN_b6);}
              if (read_eeprom(7)==2){   //8 ROJO ON
              output_high(PIN_b7);}
              output_low(PIN_a0);
                    portboff();
```

Notese que el codigo resulta extremadamente largo y repetitivo, y que ademas aun faltan por desarrollar los casos donde call==´4´ y call==´*´ y a esto agregamos que aun falta el codigo para las camas restantes.

Es por eso que se analizo que instrucciones eran las que se repetian en cada uno de los casos y de ahí, que los cuadros rojo, azul y verde que indican la rutina1(), rutina2() y portboff() se utilicen para ejemplificar el codigo.

```
}
}
   if (call =='4'){
       write_eeprom (5,2);
       if (read_eeprom(5)==2){
                           output_high(PIN_A0);
                             rutina1();                 //1-8 ROJO ON
                             output_high(PIN_c0);
                             rutina2();

}
}
   if (call =='*'){

          write_eeprom (5, 3);
                           output_high(PIN_c0);
                           rutina2();
                             output_high(PIN_a0);
                             rutina1();                 //1-8 ROJO ON
   }
         break;
} //FIN DE SWITCH
} //FIN DE LOOP();
```

Aun asi, fue necesario emplear un PIC con la suficiente memoria EEPROM para almacenar esta cantidad de datos, y por eso se eligio el PIC16F886, originalmente se utilizaba el PIC16f883 sin embargo la memoria ROM se saturaba demasiado rapido.

CAPITULO 6: PRUEBAS Y CONCLUSIONES

6.1 IMPLEMENTACIÓN FÍSICA

El elaborar el proyecto paso a paso fue fundamental para obtener un desempeño óptimo al final del mismo. Teóricamente el sistema solo consistía de dos circuitos cuya comunicación sería en forma serial y que desplegaría un valor sobre un panel del leds.

Sin embargo en la práctica, aparecieron factores que hacían que la comunicación no fuera tan fácil como se apreciaba en un inicio, tal es el caso de la longitud del conductor que ocasiono atenuación e interferencia en la transmisión de datos.

Incluso el armar el panel de leds resulto de gran ayuda porque fue útil para visualizar la forma en que debía programarse el PIC y considerar las distintas combinaciones posibles, aún más, fue importante porque a diferencia del simulador de PROTEUS, este no se pasmaba a menos que hubiese un efecto externo en los componentes electrónicos. Por estas razones resulta importante no solo quedarse enlateoría y pasar a la parte práctica ya que nos ayuda a visualizar las características reales que intervienen en el sistema.

A continuación se presenta una imagendonde aparecen los circuitos esclavo y maestro armados en protoboards junto con el cable telefónico de cien metros de longitud que fueron utilizados para realizar las pruebas

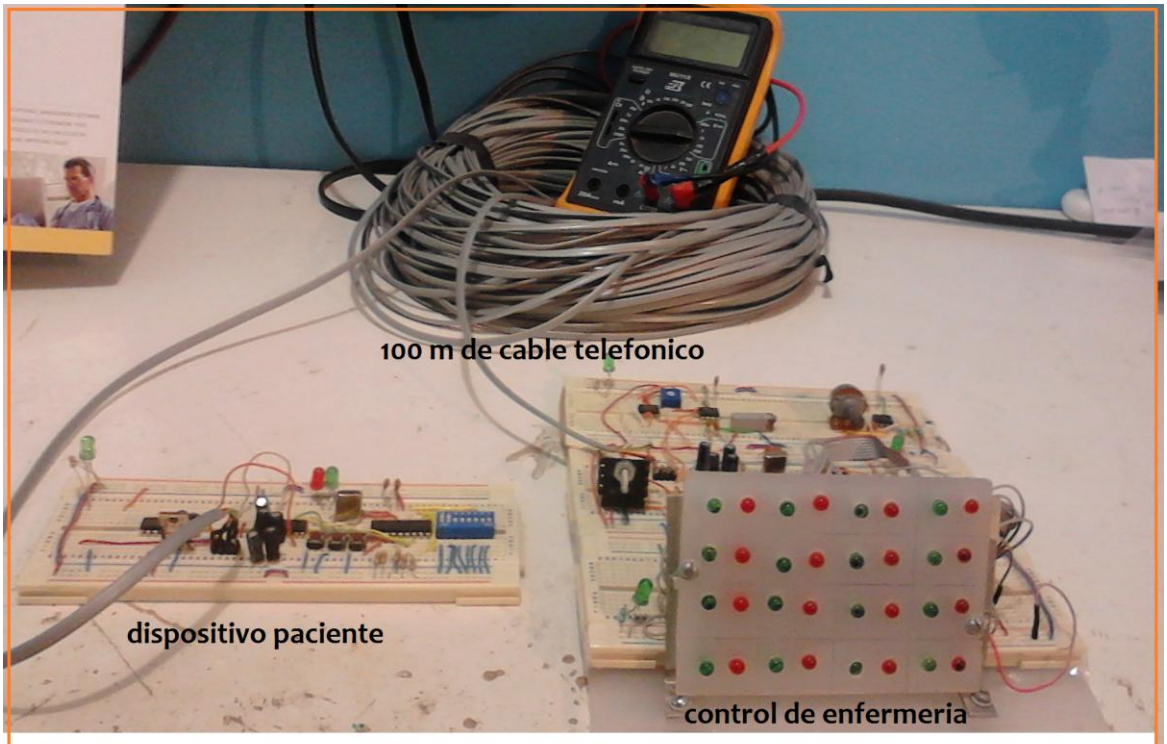


Fig. 6.1.a Sistema de comunicación paciente- personal de enfermería

Por último la Fig. 6.1.b muestra el prototipo del dispositivo que utilizara el paciente, junto con los componentes internos del mismo.Como se puede observar, el prototipo es muy sencillo, solo consta de una caja para contactos la cual es cubierta por una tapa ciega que previamente fue acondicionada para montar los botones del tipo de llamada y un par leds de alta luminosidad. En el costado superior de la caja hay un orificio por el cual entra el cable telefónico directo a un conector del tipo telefónico de cuatro vías.

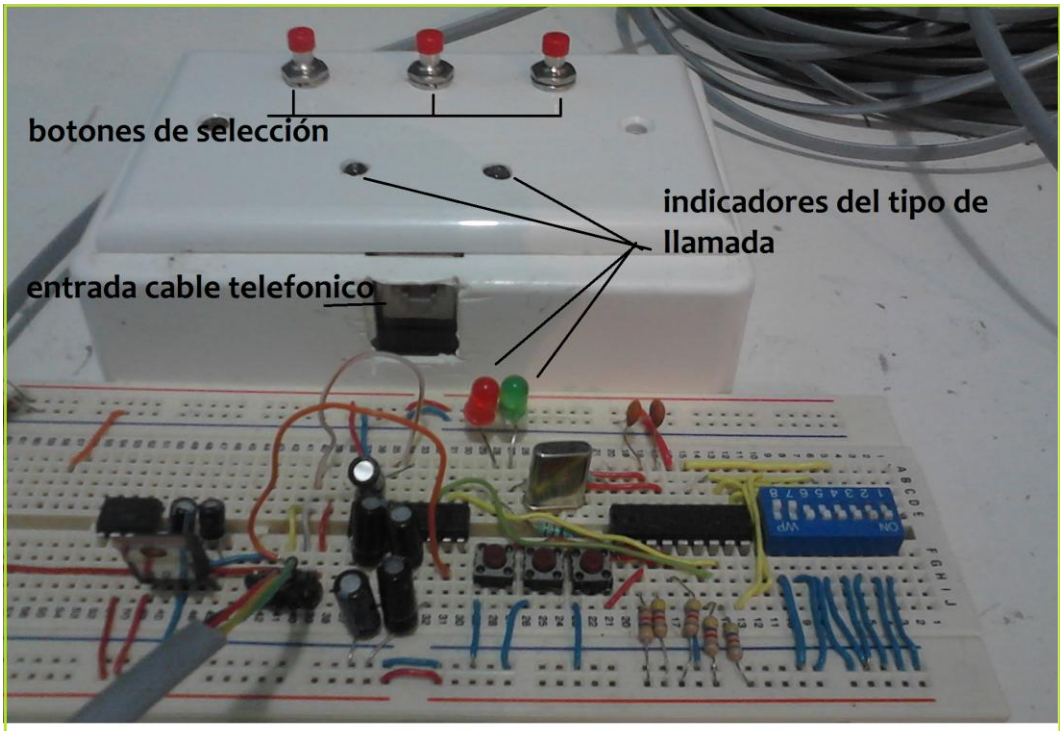


Fig. 6.1.b prototipo del dispositivo del paciente

6.2 RESULTADOS

Con esto se ha demostrado que es posible desarrollar un proyecto eficiente, económico pero sobre todo que satisface las necesidades básicas de un sistema de comunicación que comunique al personal de enfermería y derechohabientes, dicho sistema pudo desarrollarse con componentes básicos y de fácil adquisición en elmercado.

Dada mi experiencia laboral en hospitales puedo decir que la implementación de este sistema en servicios tales como: admisión continua, recuperación o quizás cubículos de hospitalización resulta practico puesto que se le ofrece al paciente la atención médicaen el momento en que lo requiere.

Sin embargo he visto que los equipos que actualmente están instalados en algunos servicios de algunos hospitales del IMSS en los que he trabajado, se encuentran fuera de servicio, principalmente porque ya son equipos obsoletos y no se les da mantenimiento. Quizás una de las razones por las que no se les da mantenimiento es porque resultaría más viable

cambiar todo el sistema que tratar de encontrar las fallas en los circuitos, en lo cantidad enorme de cableado o simplemente al intentar reponer las refacciones necesarias.

Las ventajas que ofrece mi sistema essu sencillez en caso de unmantenimiento correctivo, ya que no son muchos los componentes empleados,y el ahorro en las líneas de transmisión de datos al ser un tipo de comunicación serial.

Otras de las ventajas que ofrece es que los componentes que se utilizan no tienen que ser comprados en el extranjero ni debemos de esperar demasiado tiempo para conseguirlos. Durante el desarrollo del proyecto me encontré con un circuito que expandía la distanciadel bus I2C, este circuito es el PCA9600, dicho circuito si bien era relativamente económico tuve que esperar poco más de tres semanas para poder trabajar con él, y aun así, la calidad de las señales no era lo suficientemente buena en comparación con las señales filtradas al pasar por el MAX232 de mi diseño.

Además, al ser un diseño relativamente sencillo, los dispositivos tanto maestros como esclavos podrían ser fácilmente reparados en caso de una falla. Ya que incluso,para poder configurar el número de cama para un dispositivo maestro dado, se es suficiente con conocer las combinaciones necesarias del dip-switch para determinar el número requerido sin necesidad de reprogramar el PIC. Para esto, se podría incluso capacitar al personal de conservación perteneciente al IMSS y se evitaría tener que recurrir a terceros o contratistas, pero lo más importante sería que los mantenimientos correctivos que lleguen a requerirse se realizarían de manera inmediata.

En cuanto a costos, el modulo completo en la central de enfermería tiene un costo aproximado de 350 MXN, mientras que un módulo para paciente tiene un costo aproximado de 120 MXN, esto sería únicamente en cuanto a componentes electrónicos, ya que las cajas para cada uno de los módulos de paciente y el módulo de la central de enfermería junto con el acrílico para colocar los leds indicadores implican un costo extra.

En una primera etapa, resultaría conveniente instalar en un servicio pequeño este sistema y ver que mejoras podrían implementarse e incluso podría compararse con sistemas similares como son los diseños de *Rauland* la cuál es una empresa dedicada a este tipo de sistemas de comunicación

Por las razones antes expuestas se deduce que este proyecto resulta viable para su implementación en un hospital además de que es un diseño de manufactura mexicana.

BIBLIOGRAFÍA

- Microcontrolador PIC16F84, Desarrollo de Proyectos, Enrique Palacios, Fernando Ramiro, Lucas J. López, Edit. AlfaOmega

- **PARAMETROS DE LINEAS (OCTUBRE 2013)**

http://es.slideshare.net/josephalbertjones/parametros-de-las-lineas-electricas
http://www.forosdeelectronica.com/f16/multiplicadores-voltaje-359/

- **BUS CAN y LIN (OCTUBRE 2013)**

http://www.canbus.galeon.com/electronica/canbus.htm
https://www.youtube.com/watch?v=M1VI9wITmA4 (www.melexis.com)

- **PROTOCOLO I2C (NOVIEMBRE 2013)**

http://docs.lpcware.com/lpc800um/RegisterMaps/I2C/c-Time-out.html
https://prezi.com/fsct7p_y3okz/untitled-prezi/

- **COMUNICACIÓN ENTRE PICS (NOVIEMBRE 2013)**

http://castilloelectrinic.wordpress.com/2008/11/20/como-podemos-comunicar-PICs/

- **TRANSMISIÓN SERIE RS232, I2C y SPI, RS485 (DICIEMBRE 2013)**

http://ocw.um.es/ingenierias/sistemas-embebidos/material-de-clase-1/ssee-da-t03-01.pdf
http://server-die.alc.upv.es/asignaturas/PAEEES/2005-06/A03-A04%20-%20Bus%20CAN.pdf
http://es.scribd.com/doc/17115835/Comunicacion-Seriales-RS485
http://www.i-micro.com/pdf/articulos/RS-485.pdf
http://todoPIC.mforos.com/79706/3976971-comunicacion-RS-485-con-PIC/
http://www.epanorama.net/links/serialbus.html
http://digital.ni.com/public.nsf/allkb/039001258CEF8FB686256E0F005888D1
http://robotyPIC.blogspot.mx/2011/03/comunicacion-rs232-entre-dos-PIC.html
http://www.olimex.cl/tutorials.php?page=buses

- **EXPANSOR PARA BUS I2C (ENERO 2014)**

http://www.didacticaselectronicas.com/index.php?page=shop.product_details&flypage=flypage.tpl&product_id=143&category_id=27&option=com_virtuemart&Itemid=37&vmcchk=1&Itemid=37
www.nxp.com/documents/data_sheet/PCA9600.pdf
www.nxp.com/documents/leaflet/75016081.pdf

http://www.mouser.com/ds/0/NXP/pca9600.daughter.card-793.pdf

- **COMPILADOR CCC Y SIMULADOR PROTEUS PARA MICROCONTROLADORES PIC EDUARDO GARCIA BREIJO (ENERO 2014)**

http://books.google.com.mx/books?id=k8vMlKuRAyUC&pg=PA175&lpg=PA175&dq=funcion+I2C_START,+de+PICc&source=bl&ots=JfD5cH2EGm&sig=pAtLjr-6nmz47h8PQ8ncZoSvMKk&hl=es-419&sa=X&ei=v-BlUtHkEsqq2wWkh4GQBQ&ved=0CCkQ6AEwAA#v=onepage&q=funcion%20I2C_START%2C%20de%20PICc&f=false
http://perso.wanadoo.es/luis_ju/PIC/PIC03.html

- **CUADRO COMPARATIVO DE PROTOCOLOS (ENERO 2014)**

http://www.ucpros.com/work%20samples/Microcontroller%20Communication%20Interfaces%203.htm

- **COMPUERTAS LÓGICAS (MAYO 2014)**

https://lc.fie.umich.mx/~jfelix/LabDigI/Practicas/P6/Lab_Digital%20I-6.html
http://www.profesormolina.com.ar/electronica/componentes/int/comp_log.htm
http://www.aguilarmicros.mex.tl/imagesnew2/0/0/0/0/2/1/4/2/9/6/Comp_L.pdf

- **LATCHES Y FLIP-FLOPS (MAYO 2014)**

http://es.slideshare.net/farecrure/latches
http://mumoaldigitales1.blogspot.mx/2010/11/diferencia-entre-los-tipos-de-flip-flop.html

- **MEMORIA EEPROM (JUNIO 2014)**

http://www.aquihayapuntes.com/indice-practicas-PIC-en-c/memoria-EEPROM-interna-del-PIC.html

- **SISTEMAS LOGICOS SINCRONOS Y ASINCRONOS (MAYO 2014)**

http://www.uhu.es/diego.lopez/AI/auto_trans-tema2.pdf
http://es.scribd.com/doc/70920363/Comunicaciones-Sincronas-y-as

- **SISTEMAS COMBINACIONALES Y SISTEMAS SECUENCIALES (MAYO 2014)**

http://www.alumnos.inf.utfsm.cl/~raraya/arq/material/Capitulo_4.pdf

APENDICE

TRANSMISIÓN ASÍNCRONA

La transmisión asíncrona es aquella que no necesita de una señal de reloj sinoque transmite o recibe un carácter, añadiendo a la cadena de datos un bit de inicio y un bit de término de transferencia, para separar así los paquetes que se van enviando/recibiendo para sincronizar el receptor con el transmisor. El bit de inicio le indica al dispositivo receptor que sigue un carácter de datos; similarmente el bit de término indica que el carácter o paquete ha sido completado. (pag. 6)

TRANSMISIÓN SÍNCRONA

En este tipo de transmisión el envío de un grupo de caracteres es un flujo continuo de bits. Para lograr la sincronización del receptor y el transmisor ambos dispositivos proveen una señal de reloj (pag. 6)

COMUNICACIÓN SIMPLEX.

En un canal simplex los datos siempre viajarán en una dirección. (pag. 6)

COMUNICACIÓN HALFDUPLEX.

En un canal half duplex, los datos pueden viajar en una u otra dirección, pero sólo durante un determinado periodo de tiempo; luego la línea debe ser conmutada antes que los datos puedan viajar en la otra dirección. (pag. 6)

COMUNICACIÓN FULL DUPLEX

En un canal full duplex, los datos pueden viajar en ambos sentidos simultáneamente (pag. 6)

HANDSHAKING

"Apretón de manos" es un proceso automatizado que establecelos parámetros de comunicación entre dos dispositivos. Tienelugar cuando un equipo está a punto de comunicarse con otros dispositivos para establecer las normas de comunicación. (pag.8)

DETECCIÓN DE COLISIONES

La detección de colisiones evita daños en los datos si dos o más maestros intentan controlar el bus al mismo tiempo. Es por esto que el bus permite la conexión de varios Masters ya que incluye un detector de colisiones. (pag. 8)

CARGA ACTIVA

Una carga activa o carga dinámica es un componente de circuito que se comporta como una resistencia no lineal estable contra corriente (Pag. 10)

TIME OUT

El tiempo de espera se utilizar para detectar un bus bloqueado a fin de solucionar esta condición.(Pag. 21)

FUNCION LOGICA

Es una expresión algebraica de varias variables binarias. Dependiendo del valor que tomen las distintas variables, dicha expresión acabará tomando un valor 1 ó 0. Los sistemas que implementan funciones lógicas suelen denominarse Sistemas Combinacionales (pag.44)

a	b	$a + b$
1	1	1
1	0	1
0	0	0
0	1	1

CHARGEPUMP

Una bomba de carga es un tipo de convertidor de CC a CC que utiliza condensadores como elementos de almacenamiento de energía para crear ya sea una fuente de alimentación de voltaje más alto o más bajo.

Las bombas de carga usan el principio de cargar un condensador con la tensión de entrada, para después desconectar el condensador de la entrada y conectarlo a la salida según convenga.

Si lo que queremos el elevar la tensión de salida (Vout>Vin) lo que se hace es que con el condensador cargado se coloca en serie con la tensión de entrada, por tanto Vout = Vin + Vcondensador.

Si por el contrario queremos una tensión negativa con respecto de la entrada (-5 Vdc por ejemplo), lo que se hace es con la carga almacenada en el condensador se conecta a la salida con la polaridad invertida, por tanto Vout = - Vin. (Pag. 67)