ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE INGENIERÍA

DISEÑO Y CONSTRUCCIÓN DE UN PROTOTIPO DE LABORATORIO PARA UN SISTEMA AUTOMÁTICO DE PAGO DE PEAJES USANDO COMUNICACIONES INALÁMBRICAS MÓVILES

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

JORGE RICARDO GUATO GUAMÁN

jrguatog@hotmail.com

DIRECTOR: ING. PABLO SALINAS

psalinas58@hotmail.com

Quito, Agosto 2010

DECLARACIÓN

Yo Jorge Ricardo Guato Guamán, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Jorge Ricardo Guato Guamán

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Jorge Ricardo Guato Guamán, bajo mi supervisión.

Ing. Pablo Salinas

DIRECTOR DE PROYECTO

AGRADECIMIENTO

Agradezco a Dios por darme una gran familia que siempre me supo apoyar en los buenos y malos momentos que presenta la vida, dándome la fuerza necesaria para alcanzar cada objetivo trazado en mi vida.

Gracias al Ing. Pablo Salinas, por su acertada dirección en la culminación de este proyecto.

Jorge

DEDICATORIA

Dedico este trabajo a mi Padre Julio Guato, a mi madre Rosa Guamán por ser un ejemplo de fortaleza, dedicación, optimismo superación. A todos mis hermanos y amigos que siempre están cuando los necesito.

Jorge

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El proyecto de titulación trata del diseño y construcción de un prototipo para un sistema de pago de peajes (Telepeajes) que usa módulos inalámbricos y microcontroladores.

En el primer capítulo se analiza las principales características de funcionamiento y operatividad de los microcontroladores, memorias seriales, antenas y módulos inalámbricos. Se describen las características generales de los microcontroladores fabricados por Microchip (PICs) y ATMEL (Familia MCS-51), se realiza un estudio de las memorias seriales 24LCXX, se analiza algunos tipos de antenas y de módulos inalámbricos disponibles en el mercado.

El segundo capítulo tiene el desarrollo de los prototipos con los diagramas de los diferentes bloques que componen los módulos, se estudia al programa PCW utilizado para desarrollar los códigos fuente de los microcontroladores.

En el tercer capítulo se estudia el funcionamiento general de los sistemas electrónicos de cobro de peajes, se estudia el funcionamiento del Telepeaje del túnel de la avenida Guayasamín (Quito).

En el cuarto capítulo se indican las pruebas realizadas, los costos de los dispositivos, las conclusiones y recomendaciones.

PRESENTACIÓN

Con el incremento de vehículos que se tiene a nivel mundial, los sistemas de cobro automático cada vez tienen más importancia en las diferentes estaciones de peajes. Un cobro manual genera embotellamientos y malestar al usuario, por eso la necesidad de implementar estos sistemas de cobro.

Los telepeajes permiten a los usuarios pasar por una estación de peaje sin detener el vehículo, el cobro se lo realiza automáticamente.

El fin de este proyecto es ofrecer un sistema de cobro automático, mediante dos dispositivos, uno ubicado en la caseta de cobro y otro en el vehículo, cuando un auto se acerca a la estación de peaje los dos dispositivos se comunican, de esta manera el dispositivo de la caseta realiza el cobro al dispositivo del auto de una manera rápida, sin que se detenga el vehículo.

El presente diseño utiliza los microcontroladores PIC16F876A-I/SO y PIC18F4550-I/PT que sirve para cumplir con las necesidades de procesamiento de datos. Estos microcontroladores fueron escogidos porque se encuentra fácilmente en el mercado, su fácil instalación y programación.

CAPÍTULO 1

1. INTRODUCCIÓN

1.1 GENERALIDADES

Las comunicaciones inalámbricas por RF entre dispositivos electrónicos hoy en día tienen gran importancia en las transmisiones de datos, principalmente porque no requieren línea de vista para comunicarse, teniendo una infinidad de aplicaciones.

El objetivo de este proyecto es diseñar un prototipo para un sistema electrónico de pago de peaje mediante el uso de comunicaciones inalámbricas. Con su implementación no se tendrá que manejar dinero en efectivo en el pago de peaje, facilitando el control de las diferentes empresas que tengan autos para sus empleados, o de las compañías de transporte público que tengan que pasar por una caseta de pago de peaje varias veces al día, se aumentará la fluidez de vehículos porque estos no se detienen ya que el pago se lo hará en forma automática.

Al circuito a instalarse en el vehículo se lo llama Dispositivo móvil (DIMOVIL) y el que recibe los datos en la caseta se lo llama Dispositivo fijo (DIFIJO).

1.2 MICROCONTROLADORES[1]

Un Microcontrolador es un circuito integrado que contiene muchas de las mismas cualidades que una computadora de escritorio, tales como la CPU (Unidad Central de Procesamiento), memoria RAM (Memoria de Acceso Aleatorio), EEPROM (ROM

[1] http://www.olimex.cl/tutorial/tutorial1.pdf

programable y borrable eléctricamente) y circuitos de entrada y salida. En la Figura 1.1 se muestra los componentes internos de un microcontrolador. Pero no incluye ningún dispositivo de comunicación con humanos, como monitor, teclado, o mouse.

Diseñados para aplicación de control de máquinas, más que para interactuar con humanos.

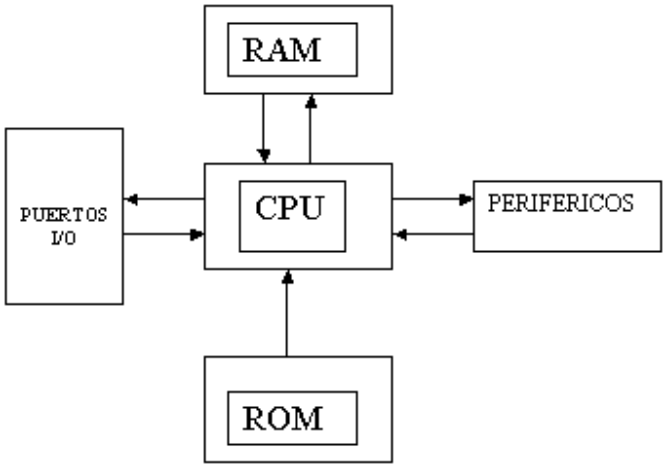


Figura 1.1 Componentes internos de un Microcontrolador[2]

Los microcontroladores son dispositivos electrónicos capaces de realizar procesos lógicos, estos procesos o acciones deben ser programados en lenguaje ensamblador por el usuario y son introducidos en este a través de un programador. En su memoria reside solo el programa que controla una aplicación determinada. Son muy versátiles, ya que con el mismo dispositivo se puede controlar el parpadeo de LEDs, un horno de microondas e incluso realizar la automatización de una fábrica.

Se puede tener una infinidad de aplicaciones, todo depende de la imaginación del usuario y de la capacidad de programación.

[2] http://www.aiu.edu/applications/DocumentLibraryManager/upload/Despradel%20Novas%20Pe%C3%B1a.pdf

Debido a que puede reemplazar muchos circuitos lógicos tales como las compuertas, conversores A/D (Analógico/Digital), D/A (Digital Analógico), decodificadores, etc. Se tiene un gran ahorro de espacio en las placas.

En definitiva un microcontrolador es un sistema cerrado, todas las partes del procesador están contenidas en su interior y solo salen al exterior las líneas que controlan los periféricos. En la Figura 1.2 se tiene un gráfico del microcontrolador.

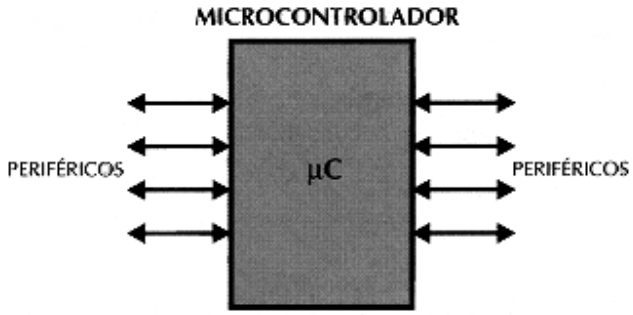


Figura 1.2 Microcontrolador

Existen varios fabricantes de microcontroladores tales como Texas Instruments, Motorola, Atmel, Intel, Microchip, Toshiba, etc. Todos ellos ofrecen microcontroladores con características más o menos similares, sin embargo, en términos generales se puede decir que todos sirven para lo mismo: leer y ejecutar los programas del usuario. Evidentemente algunos modelos tienen más capacidad que otros, en cuanto a memoria, velocidad, periféricos, etc.

1.2.1 VENTAJA DE LOS CIRCUITOS CON MICROCONTROLADORES

- Se tiene una gran reducción del tamaño del circuito y del precio del proyecto. El elevado grado de integración de un microcontrolador en un circuito

integrado permite una elevada funcionalidad por área a bajo coste y un menor tamaño del PCB[3].

- Elevada flexibilidad, un mismo microcontrolador puede ser usado por un elevado número de aplicaciones variando solo el software.
- Rapidez de desarrollo, la adaptación de un microcontrolador en otra aplicación puede consistir en adaptar el software y muy poco hardware.
- Aumento de la fiabilidad, la disminución de componentes en placa hace disminuir también los riesgos de averías.
- Buenas prestaciones, los microcontroladores usan microprocesadores que permiten la ejecución eficiente de algoritmos de control.

1.2.2 ARQUITECTURA BÁSICA DE LOS MICROCONTROLADORES

En esta sección se enumeran los elementos que tienen la mayoría de los microcontroladores, todo depende de la versión de microcontrolador que se escoja según las necesidades.

1.2.2.1 Unidad Central de Procesamiento o CPU[4]

La CPU es el microprocesador del sistema, determina sus principales características, tanto a nivel de software como hardware.

Sus características y funcionalidad se definen a partir de la siguiente clasificación:

[3] Circuito impreso (del inglés Printed Circuit Board)
[4] http://www.ibars.com/assets/presentations/PIC18%20(sph).pdf

- En función del tamaño de los datos
    - 4 Bits: Aplicaciones muy sencillas y muy económicas
    - 8 Bits: Aplicaciones sencillas y son económicos.
    - 16 Bits: Aplicaciones medias y costo relativamente medio
    - 32 Bits: Aplicaciones complejas y coste elevado

- En función del conjunto de instrucciones
    - RISC (Reduced Instruction Set Code). Instrucciones sencillas y de rápida ejecución.
    - CISC (Complex Instruction Set Code). Instrucciones más complejas y de mayor tiempo de ejecución.

- En función de la arquitectura de los buses
    - Von Newmann: Buses de datos y direcciones compartidos por la memoria de datos y de programa. Simplifica el diseño y el costo del microcontrolador.

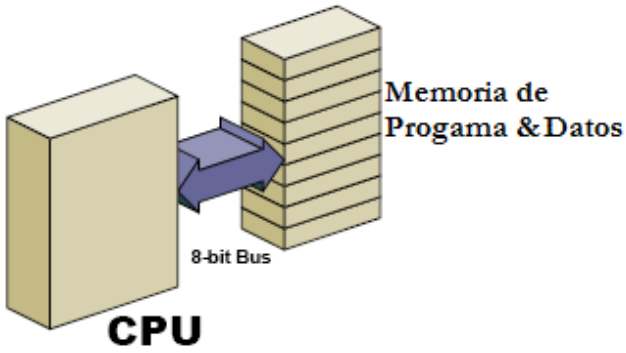


Figura 1.3 Arquitectura Von Newmann[5]

[5] http://www.ibars.com/assets/presentations/PIC18%20(sph).pdf

- Harvard: El término proviene de la computadora Harvard que almacenaba las instrucciones en cintas perforadas y los datos en interruptores. Esta arquitectura cuenta con la memoria de programa y la memoria de datos separado y solo accesibles a través de buses distintos. Esta arquitectura ofrece la posibilidad de poder acceder a una sola instrucción en un ciclo de reloj. Mientras la memoria de programa es accedida, la memoria de datos esta en un bus independiente y puede ser leída y escrita. La figura muestra un diagrama de bloques de la arquitectura Harvard.

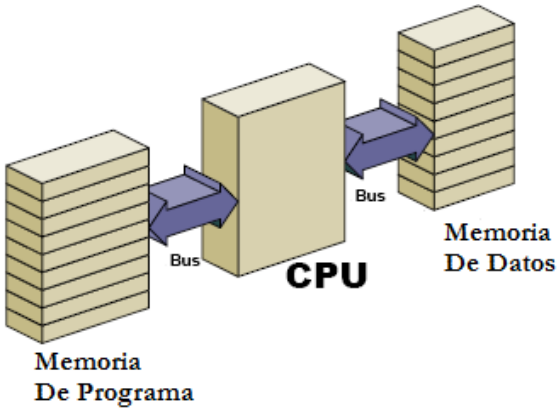


Figura 1.4 Arquitectura Harvard[6]

1.2.2.2 Memoria de programa

El programa que se escribe se guarda en esta memoria. Es una memoria que se mantiene aunque se apague el sistema que lo contiene. En función de cómo se borra o como se graba se tiene la siguiente clasificación:

- ROM: Memoria sólo de lectura, su contenido se graba durante la fabricación del chip.

[6] http://www.ibars.com/assets/presentations/PIC18%20(sph).pdf

- EPROM: Pueden borrarse y grabarse muchas veces. La grabación se realiza con un grabador controlado desde un PC. Si, posteriormente, se desea borrar el contenido, disponen de una ventana de cristal en su superficie por la que se somete a la EPROM a rayos ultravioleta durante varios minutos
- EEPROM: Tanto la programación como el borrado, se realizan eléctricamente desde el propio grabador y bajo el control programado de un PC
- FLASH: Es una forma desarrollada de la memoria EEPROM que permite que múltiples posiciones de memoria sean escritas o borradas en una misma operación de programación mediante impulsos eléctricos.

1.2.2.3 Memoria de datos

Dependiendo de si se mantiene o no cuando se apaga el sistema se tiene la siguiente clasificación:

- Memoria Volátil de Datos: En esta memoria se guarda temporalmente variables usadas en el programa, hay una genérica y otra más específica:
    - RAM (Memoria de Acceso Aleatorio): Almacena las variables del programa
    - SFR: (Registros de función especial), configuran a todo el microcontrolador, con estos registros se controla que un pin (pórtico) sea de entrada o salida, se controla las interrupciones, se inicia o se pausa un Timer, se ve la información proveniente de una PC a través del puerto serie.

- Memoria No Volátil De Datos: Es una memoria como la anterior que sirve para almacenar datos, pero en este tipo de memoria se mantiene, aún cuando se desconecta la fuente de energía.

1.2.2.4 Temporizadores o Timers

Se llama Timer al contador de ciclos de una señal de reloj generado internamente en el dispositivo, se utiliza para controlar periodos de tiempo (temporizadores). Se carga en un registro un valor y a continuación dicho valor se va incrementando o decrementando al ritmo de los impulsos del reloj o algún múltiplo hasta que se desborde y llegue a cero, momento en que se produce un aviso.

Se utiliza también para llevar la cuenta de acontecimientos que suceden en el exterior (contadores). El conteo se produce por cambios de nivel o flancos en alguno de las E/S del microcontrolador, el registro se va incrementando o decrementando al ritmo de dichos impulsos.

1.2.2.5 Perro Guardián o WatchDog

Consiste en un temporizador descendente que puede ser utilizado para recuperar el control del microcontrolador cuando el sistema se queda colgado, su función es realizar un reset periódicamente para refrescarlo. Esto evita que cuando el programa ingrese a alguna parte y no pueda salir se quede indefinidamente en este ciclo.

El usuario puede realizar la activación del “WatchDog” por software o hardware. Si se selecciona la opción hardware el WatchDog se inicializa inmediatamente después del reset.

1.2.2.6 Protección ante fallo de alimentación o Brown-Out

El Brown-Out es un circuito interno que sirve como detector de posibles errores en la alimentación del microcontrolador, paralizando el sistema para evitar posibles daños en éste. Este circuito realiza un reset al microcontrolador cuando el voltaje de alimentación es inferior al voltaje mínimo (Brown-Out). Mientras el voltaje sea inferior al Brown-Out el dispositivo no se reinicia, comenzando a funcionar normalmente cuando sobrepase dicho valor.

1.2.2.7 Estado de reposo o bajo consumo

Los microcontroladores para ahorrar energía disponen de una instrucción que les pasa al estado de reposo (estado de inactividad o idle) o bajo consumo, en el cual los requerimientos de potencia son mínimos. El microcontrolador entra en funcionamiento normal cuando es recibida una interrupción ocasionada por un evento esperado o se produce un reset por hardware.

1.2.2.8 Conversores A/D y D/A

Los procesadores que incorporan conversores A/D pueden procesar señales analógicas, mientras que los conversores D/A transforman los datos digitales en su correspondiente señal analógica.

1.2.2.9 Puertos de entrada y salida

Son los pórticos (pins) destinados a soportar líneas de entrada/salida digitales, por lo general estas líneas se agrupan de ocho en ocho, formando puertos

1.2.2.10 Puertos de comunicación

Los microcontroladores son capaces de comunicarse con otros dispositivos externos mediante un puerto de comunicación serial, dependiendo el tipo de microcontrolador, éstos pueden ser:

- I^2C (Inter-Integrated Circuit)
- UART (Universal Asynchronous Receiver-Transmiter)
- USART (Universal Synchronous-Asynchronous Receiver-Transmiter)
- CAN (Controller Area Network)
- USB (Universal Serial Bus)

1.2.2.11 Modulador de anchura de pulso o PWM

El PWM es un generador de pulsos de anchura variable, útil para controles del mismo tipo haciendo la función de duty-cycle[7], es decir, variaciones en las duraciones de los semiciclos positivos y negativos para obtener tensiones medias variables y conseguir así, por ejemplo, controlar la velocidad de un motor DC.

También se puede variar la frecuencia de estos pulsos y conseguir así, por ejemplo, emitir diferentes sonidos en una aplicación.

1.2.3 MICROCONTROLADORES PIC[8]

Los microcontroladores PIC son fabricados por la empresa Microchip que tiene una gran variedad de modelos de los cuales se puede seleccionar uno de acuerdo a

[7] Fracción de tiempo que el sistema permanece en estado activo
[8] http://cde05.etse.urv.es/pub/pdf/524pub.pdf

nuestras necesidades. Tiene una gran variedad de herramienta para desarrollar hardware y software.

1.2.3.1 Características de los PIC

Entre sus características de tiene:

- RISC (Reduced Instruction Code)
    - Gama baja (PIC 16C5X) 33 instrucciones
    - Gama media (PIC 16CXXX) 35 instrucciones
    - Gama alta (PIC 17CXXX/18CXXX) 58/77 instrucciones
- Arquitectura Harvard
    - Memoria de datos de 8 bits
    - Memoria de programa 12/14/16 bits
- Arquitectura Pipeline.- Consiste en ir transformando un flujo de datos en un proceso comprendido por varias fases secuenciales, siendo la entrada de cada una la salida de la anterior.
    - Todas las instrucciones ocupan una palabra de instrucción
    - Ejecución de todas las instrucciones en dos ciclos
- Pila Hardware
- WatchDog (WDT)
- Power On Reset (POR).- El pulso de inicialización POR (Power-On Reset) es generado internamente cuando la tensión de alimentación se encuentra entre

los limites de 1.2 V y 1.7 V, es un método cómodo de originar un reset al conectar la alimentación, ya que no requiere de ningún componente adicional.

- Modo de bajo consumo (SLEEP)
- Líneas de E/S de alta corriente (20/25 mA)
- Protección de código
- Número de serie/código de identificación
- Memoria de programación:
    - C = CMOS OTP/EPROM
    - CR = CMOS ROM
    - CE = CMOS OTP/EPROM+EEPROM
    - F = FLASH

1.2.3.2 Familia de los PIC

- Familia PIC 16C5X
    - Palabra de programación de 12 bits
    - Familia base
    - 2 niveles de pila de hardware
    - Sin interrupciones
    - 1 Timer de 8 bits + WDT
- Familia PIC 12C5XXX

  - Palabras de programación de 12 y 14 bits
  - EEPROM
  - Interrupciones
  - 1 Timer de 8 bits + WDT
- Familia PIC 16CXXX, 16FXXX
  - Palabra de programación de 14 bits
  - Prestaciones medias
  - Gran variedad de periféricos incluidos en el circuito integrado (on-chip): Comparadores, PWM, 3 Timers, Conversores A/D, EEPROM de datos, USART.
  - 8 niveles de pila de hardware
  - Interrupciones internas y externas
- Familia PIC 17CXXX
  - Palabra de programación de 14 bits
  - Otras prestaciones
  - Gran variedad de periféricos incluidos en el circuito integrado (on-chip)
  - 16 niveles de pila de hardware
  - Interrupciones vectorizadas
- Familia PIC 18CXXX
  - Palabra de programación de 16 bits
  - Muchas otras prestaciones (10 MIPS)

- Gran variedad de periféricos on-chip: Comparadores, PWM, 3 Timers, Conversores A/D, EEPROM de datos, USART.
- 32 niveles de pila de hardware

1.2.4 MICROCONTROLADRES ATMEL[9]

ATMEL fabrica microcontroladores de la familia MCS-51 y de la familia AVR, esta nueva tecnología proporciona todos los beneficios habituales de la arquitectura RISC y memoria FLASH programable eléctricamente. La característica que los identifica a estos microcontroladores de ATMEL es la memoria FLASH y EEPROM que incorpora.

El diseño AVR de ATMEL difiere de los demás microcontroladores de 8 bits por tener mayor cantidad de registros (32) y un conjunto ortogonal de instrucciones.

El gran conjunto de registros disminuye la dependencia respecto a la memoria, lo cual mejora la velocidad y disminuye las necesidades de almacenamiento de datos. Además casi todas las instrucciones se ejecutan en 1 ó 2 ciclos de reloj. Adicionalmente, ATMEL también proporciona en línea el entorno software (AVR STUDIO) que permite editar, ensamblar y simular el código fuente

1.2.4.1 Comunicación de los microcontroladores ATMEL[10]

Los microcontroladores ATMEL tienen los registros de función especial (SFR), los cuales se observa en la tabla 1.1. Estos registros se encuentran en la RAM interna excepto el registro contador de programa y los 4 bancos de registros.

[9] http://www.atmel.com/dyn/products/datasheets.asp?family_id=604
[10] PDF: Atmel 8051 Microcontrollers Hardware Manual

Ocupan las direcciones desde 80H hasta FFH (128 bytes altos) y son accedidos con direccionamiento directo. Algunos de estos registros poseen bits que son direccionables independientemente.

En el bloque de memoria designado para los SFR, no todas las localidades son asignadas, se han dejado, localidades libres para futuros modelos de microcontroladores.

F8							.....		FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8	(T2CON)[11]	(T2MOD)	(RCAP2L)	(RCAP2H)	(TL2)	(TH2)			CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87

Tabla 1.1 Mapa de los SFR

1.2.4.2 Interface serial[12]

Para acceder a la recepción y transmisión desde y hacia el microcontrolador se lo puede hacer a través del registro SBUF (Buffer de datos del pórtico serial), este sirve para las dos cosas, transmitir datos y recibir datos y en ambos modos trabaja en forma diferente.

[11] Presente en los microcontroladores ATMEL serie 52
[12] PDF: Atmel 8051 Microcontrollers Hardware Manual

Entre las características del puerto serial se tiene que es full-dúplex con lo cual se puede transmitir y recibir simultáneamente. También tiene la característica de receive-buffered, esto significa que puede empezar a recibir un segundo byte aún cuando el byte previo no ha sido leído del registro de recepción. Sin embargo si el primer byte no ha sido leído cuando la recepción del segundo byte este completo uno de ellos se perderá.

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Tabla 1.2 SCON (Registro De Control Del Pórtico Serial)

Símbolo	**Posición**	**Nombre y Significado**
SM0	SCON.7	Especifica el modo de operación del pórtico serial
SM1	SCON.6	
SM2	SCON.5	Habilita la comunicación de multiprocesador, característica en los modos dos y tres. Si SM2=1, entonces RI no se activará si el 9^{no} bit de datos (RB8) es 0. En el modo 1, si SM2=1, entonces RI no se activará si es que no se recibe un bit de parada válido. En el modo 0, SM2 debería ser 0
REN	SCON.4	Seteado/borrado por software para habilitar / deshabilitar la Recepción.
TB8	SCON.3	Es el 9^{no} bit de datos que es transmitido en los modos 2 y 3. Seteado o borrado por software.
RB8	SCON.2	En los modos 2 y 3, contiene el 9^{no} bit de datos recibido. En el modo 1, si SM2=0, RB8 es el bit de parada recibido. En el modo 0, RB8 no se utiliza
TI	SCON.1	Flag de interrupción del transmisor. Se activa al final del 8^{vo} bit en Modo 0, o al inicio del bit de “PARADA” en los otros modos. Debe borrarse por programa
RI	SCON.0	Flag de interrupción del receptor. Se activa al final del 8^{vo} bit en Modo 0, o dentro del bit de “PARADA” en los otros modos (salvo modo multiprocesador). Debe borrarse por programa

Tabla 1.3 Registro SCON

El SFR que controla al puerto serial es el SCON (Registro De Control Del Pórtico Serial), y es direccionable a nivel de bit (tabla 1.2), en la tabla 1.3 se describe los bits de este Registro.

Los bits 6 y 7 son para la configuración del modo de operación y la velocidad, en los microcontroladores de la familia 51 se tiene 4 modos del puerto serie y se indican a continuación.

MODO	SM0	SM1	Descripción	Velocidad de Comunicación
0	0	0	Registro de desplazamiento	$f_{osc}/12$
1	0	1	[13]UART de 8 bits	Variable
2	1	0	UART de 9 bits	$f_{osc}/64$ ó $f_{osc}/32$
3	1	1	UART de 9 bits	Variable

Tabla 1.4 Modos De Operación Del Pórtico Serial

a) Modo 0

En este modo, 8 bits de datos son Transmitidos/Recibidos. La velocidad es fijada en 1/12 de la frecuencia de oscilación (f_{osc}).

b) Modo 1

En este modo 10 bits son Transmitidos/Recibidos: 1 bit de inicio (0_L), 8 bits de datos y 1 bit de parada (1_L). En recepción el bit de parada es el RB8 del registro de función especial SCON. La velocidad es variable.

c) Modo 2

[13] UART Transmisión Recepción Asincrónico Universal

En este modo 11 bits son Transmitidos/Recibidos: 1 bit de inicio (0_L), 8 bits de datos, 1 bit de datos programable y un bit de parada (1_L).

El noveno bit se carga en TB8 y RB8 para la transmisión y recepción respectivamente. Así por ejemplo en la transmisión puede cargarse el bit de paridad P que se carga en el registro PSW. En recepción este bit de parada es ignorado. La velocidad de comunicación en este modo puede configurarse para trabajar en 1/64 ó 1/32 de la frecuencia de oscilación del reloj del microcontrolador.

d) Modo 3

El modo 3 es el mismo que el modo 2 excepto que la velocidad es variable. En estos 4 modos, la transmisión es inicializada por una sola instrucción que usa al SBUF como un registro destino. En el modo 0 la recepción es inicializada por la siguiente condición: RI=0 y REN=1 y en los otros modos cuando llega el bit de inicio si REN=1.

1.2.4.3 Velocidad de transmisión[14]

Al analizar la velocidad del microcontrolador, en el registro SCON, también se puede diferenciar las velocidades, en cada uno de los modos, esto se describe a continuación.

a) Modo 0

En este modo la velocidad de comunicación es en baudios y está dada por la frecuencia de oscilación del microcontrolador y se expresa en la siguiente fórmula.

[14] PDF: Atmel 8051 Microcontrollers Hardware Manual

Velocidad (Baudios en Modo 0) = f_{osc} / 12

b) Modos 1 y 3

La velocidad de estos modos está determinada por la relación de overflow de los Timers 1 o 2 o con ambos. Al usar el Timer 1, los baudios se obtienen por el valor de carga y de overflow del registro contador del Timer y por el valor del bit SMOD (bit 7 del registro PCON) de acuerdo a la siguiente fórmula.

$$\text{Velocidad (Baudios en Modos 1 y 3)} = \frac{2^{\text{SMOD}}}{32} * \textit{Overflow del Timer 1}$$

Si es que el Timer 1 se configura como temporizador en modo de autocarga, la velocidad de comunicación está dado por:

$$\text{Velocidad (Modos 1 y 3)} = \frac{2^{\text{SMOD}} * \textit{Fecuencia de oscilación}}{32 * 12 * (256 - \textit{TH1})}$$

Si se quiere utilizar el Timer 2 como generador de baudios se debe configurar los bits TCLK y RCLK del registro T2CON con 1_L, de esta forma se consigue variar la velocidad de la transmisión y recepción según se activen TCLK y/o RCLK respectivamente. La relación del cálculo está dada por la siguiente expresión:

$$\text{Velocidad (Baudios en Modos 1 y 3)} = \frac{\text{Relación de overflow de T2}}{16}$$

Cuando el Timer 2 trabaja como temporizador común, el registro de conteo se incrementa con cada ciclo de máquina y cuando trabaja como generador de baudios se incrementa con cada periodo de estado, es decir a ½ de la frecuencia del

oscilador del microcontrolador. La relación de cálculo está dada por la siguiente expresión:

$$\text{Velocidad (Baudios en Modos 1 y 3)} = \frac{\text{Frecuencia de oscilación}}{32 * (65536 - (RCAP2H, RCAP2L))}$$

c) Modo 2

La velocidad de transmisión en el Modo 2, depende del bit SMOD del registro PCON. Si SMOD = 0, entonces la velocidad de transmisión es 1/64 y si SMOD = 1, la velocidad de transmisión es 1/32 de la frecuencia de oscilación del microcontrolador esto se visualiza en la siguiente expresión:

$$\text{Velocidad (Modos 1 y 3)} = \frac{2^{\text{SMOD}} * Fecuencia\ de\ oscilación}{64\ /\ 32}$$

En los AT89C51, el overflow del Timer 1 determina la velocidad de transmisión en los modos 1 y 3. En los AT89C52, la velocidad de transmisión puede ser determinada por el Timer 1, Timer 2 o ambos (una para transmisión y el otro para recepción). En la tabla 1.5 se lista los valores de la velocidad de transmisión comúnmente utilizados y como pueden ser obtenidos del Timer 1.

Velocidad de Transmisión	f_{osc}	SMOD	Timer 1		
			C/T	Modo	Valor Cargado
Modo 0 Max: 1 MHz	12 MHz	X	X	X	X
Modo 2 Max: 375 K	12 MHz	1	X	X	X
Modo 1, 3: 62.5 K	12 MHz	1	0	2	FFH
19.2K	11.059 MHz	1	0	2	FDH
9.6K	11.059 MHz	0	0	2	FDH
4.8K	11.059 MHz	0	0	2	FAH
2.4K	11.059 MHz	0	0	2	F4H
1.2K	11.059 MHz	0	0	2	E8H
137.5	11.059 MHz	0	0	2	1DH
110	6 MHz	0	0	2	72H
110	12 MHz	0	0	1	FEEBH

Tabla 1.5 Velocidad Generadas por el Timer 1

1.2.4.4 Familia de los ATMEL[15]

La tabla 1.6 muestra los modelos de microcontroladores más comunes de la familia ATMEL y la tabla 1.7 muestra los microcontroladores denominados AVRs.

Nombre	Memoria de Programa	Memoria de Datos	Timers	Tecnología
AT89C1051U	1 Kbyte	64 Bytes	2	CMOS
AT89C2051	2 Kbyte	128 Bytes	2	CMOS
AT89C4051	4 Kbyte	128 Bytes	2	CMOS
AT89C51	4 Kbyte	128 Bytes	2	CMOS
AT89C52	8 Kbyte	256 Bytes	3	CMOS
AT89S51	4 Kbyte	128 Bytes	2	CMOS
AT89S52	8 Kbyte	256 Bytes	3	CMOS
AT89S53	12 Kbyte	256 Bytes	3	CMOS
AT89S8252	8 Kbyte	256 Bytes	3	CMOS

Tabla 1.6 Microcontroladores ATMEL

Nombre	FLASH (Kbytes)	EEPROM (Bytes)	SRAM (Bytes)
AT90S2313	2	128	128
AT90S8515	8	512	512
AT90S8535	8	512	512
AT90S1200	1	64	-
ATmega8	8	512	1 K
ATtiny26	2	128	128

Tabla 1.7 AVRs

[15] http://www.atmel.com/dyn/products/datasheets.asp?family_id=604

1.3 COMUNICACIÓN SERIAL[16]

La comunicación serie es muy importante en el campo de los microcontroladores, ya que de esta forma deja de ser un chip aislado y le permite abrirse al mundo, interactuando con cualquier dispositivo que también soporte comunicación serial.

Para que dos dispositivos se puedan comunicar tienen que ponerse de acuerdo en que sistema lo van hacer. La comunicación serie se divide en dos categorías:

- Comunicación serial síncrona
- Comunicación serial asíncrona

La síncrona requiere de un reloj y una línea de datos. Los datos se van transmitiendo uno a uno con cada pulso de reloj. Los caracteres se transmiten consecutivamente, no existe bit de inicio ni bit de parada entre estos. De este tipo de comunicación es el protocolo I^2C (Inter Integrated Circuit).

En ***la asíncrona*** no existe una línea de reloj común que establezca la duración de un bit y el caracter puede ser enviado en cualquier momento. Esto conlleva que cada dispositivo tiene su propio reloj y que previamente se ha acordado que ambos dispositivos transmitirán datos a la misma velocidad.

En este proyecto se utiliza la memoria serial 24LC08B, que es compatible con el protocolo I^2C.

[16] http://www.iua.upf.es/~jlozano/interfaces/interfaces6.html
http://www.ate.uniovi.es/fernando/Doc2003/SED/SCI_sincrono.pdf

1.3.1 PROTOCOLO I²C (INTER INTEGRATED CIRCUIT)[17]

Es un protocolo de comunicaciones desarrollado por Philips, la principal utilidad es la de comunicación entre periféricos, cuando la distancia no es muy grande. Este protocolo define:

- Tipo de bus
- Maestros y esclavos

1.3.1.1 Tipo de bus

El protocolo define un tipo de bus serie que utiliza dos hilos trenzados para interconectar distintos periféricos. Una línea se utiliza para la señal de reloj (SCL[18]) y la otra línea para datos (SDA[19]), Las líneas SCL y SDA están conectadas a una fuente de poder por medio de resistencias de pull-up, y su valor está comprendido entre 1KΩ y 10KΩ dependiendo de la tensión de alimentación y de los dispositivos conectados. Por ejemplo, para una tensión de alimentación de 5V se tiene una resistencia de 2 KΩ.

Con el mismo circuito (2 hilos) se puede llegar a controlar hasta 128 dispositivos. Es orientado exclusivamente a aplicaciones de 8 bits, controladas por microprocesador (al menos uno como maestro). Los periféricos pueden ser:

- Drivers para LCDs
- Drivers para LEDs
- Conversores de datos

[17] http://www.nxp.com/documents/application_note/AN10216.pdf
http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf
[18] Serial Clock
[19] Serial Data

- Memorias seriales
- Relojes
- Microcontroladores

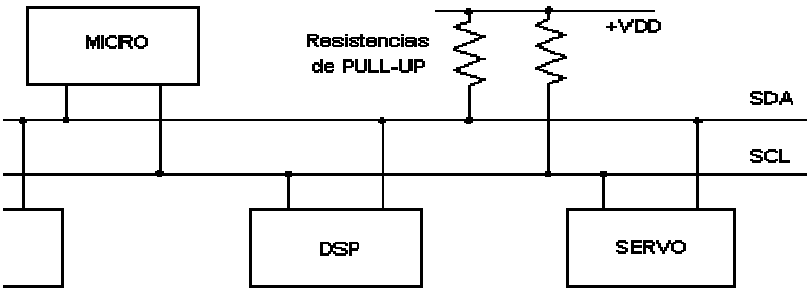


Figura 1.5 Bus I^2C[20]

1.3.1.2 Maestros y esclavos

El bus I^2C permite la conexión de varios maestros ya que incluye un detector de colisiones. El maestro (master) es el encargado de iniciar y terminar la transferencia de información y es el que genera la señal de reloj, cuya frecuencia es fija. Los esclavos (slave), hasta 128 son los dispositivos direccionados por el maestro, mediante 7 bits (dirección del esclavo).

La línea de datos es utilizada tanto por el maestro como por el esclavo para la transmisión de información. La transmisión de datos puede ser:

- Del maestro al esclavo (escritura)
- Del esclavo al maestro (lectura)

[20] http://www.comunidadelectronicos.com/articulos/i2c.htm

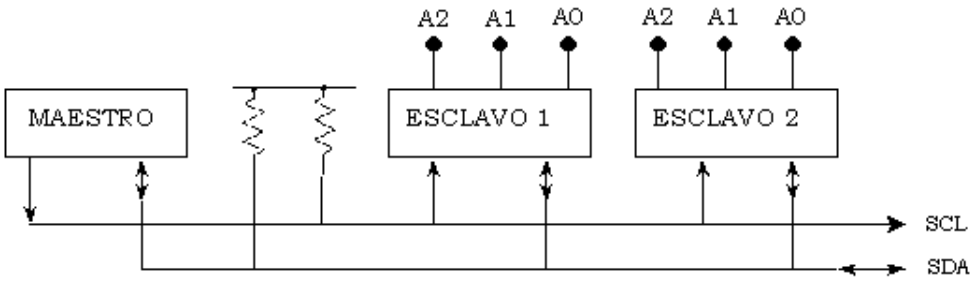


Figura 1.6 Maestros Y Esclavos[21]

Cuando el maestro inicia una trama de comunicación, envía a través de la línea de datos la dirección del esclavo con el que se pretende establecer una comunicación. Todos los esclavos reciben dicha dirección, pero solo uno responde y el resto permanece en espera de que se inicie una nueva trama.

Las resistencias de pull-up tiene como objetivo asegurar una mínima carga de la línea, necesaria para que la transmisión sea estable ante posibles "ruidos" externos y permitir que tanto la línea de datos como la de reloj sean bidireccionales, ya que cualquier dispositivo (maestro o esclavo) puede forzar a nivel bajo la línea o dejarla libre, teniendo entonces un nivel alto en la línea, siendo leída por los otros dispositivos, esto es fundamental cuando el sistema tiene conectado más de un maestro.

Cuando la línea de datos y la de reloj están a nivel alto (ningún dispositivo está actuando), se dice que el bus está en "reposo" o "libre".

Según la velocidad de transmisión se tiene los siguientes tipos de transferencia:

- Modo Estándar: 100 Kbps
- Modo rápido: 400 Kbps
- Modo alta velocidad: 3.4 Mbps

[21] http://www.comunidadelectronicos.com/articulos/i2c.htm

1.3.2 ESPECIFICACIONES DE LA COMUNICACIÓN[22]

Antes de que se establezca un intercambio de datos entre el circuito Maestro y los esclavos, el Maestro debe informar el comienzo de la comunicación (condición de start): la línea SDA cae a cero mientras SCL permanece en nivel alto. A partir de este momento comienza la transferencia de datos. Una vez finalizada la comunicación se debe informar de esta situación (condición de Parada): la línea SDA pasa a nivel alto mientras SCL permanece en estado alto.

En la figura 1.7 se representa los niveles de señal con los que funciona el bus I^2C.

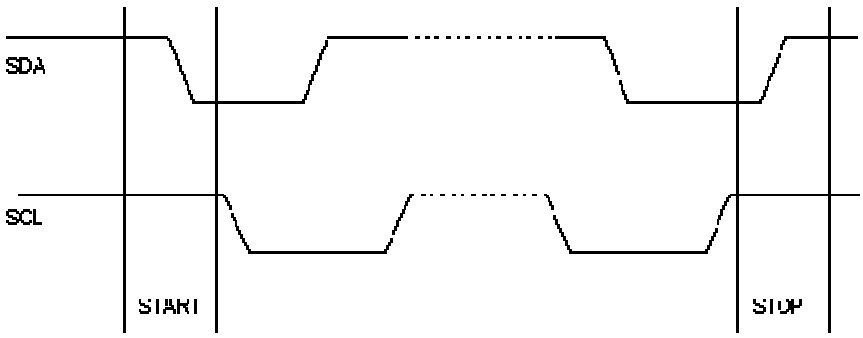


Figura 1.7 Niveles de Señal Con Los Que Funciona El Bus[23]

En el formato I^2C cada dispositivo tiene asignado una dirección de 7 bits, que envía el maestro cuando comienza una trama de comunicación con uno de ellos.

El primer byte después de la condición de INICIO determina el esclavo direccionado por el maestro. En toda trama se tiene el bit de inicio (S) y una dirección del periférico o esclavo, tras la dirección del esclavo se añade el bit de recepción/transmisión o lectura/escritura (R/W).

[22] http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf
http://www.nxp.com/documents/application_note/AN10216.pdf
[23] http://www.comunidadelectronicos.com/articulos/i2c.htm

A continuación del bit R/W, se envía un bit de acuse de recepción o reconocimiento (ACK). Tras todos estos bits, se transmiten los datos, en grupos de 8 bits cada uno, al final de cada octeto se inserta un bit de acuse de recepción, que en este caso puede ser producido por el esclavo o por el maestro. Estos datos pueden tener una función especial dentro de la trama según el orden que tengan, por esta razón depende del formato que tenga esta transmisión, o sea, depende de los dispositivos. Para terminar la trama el maestro generará el bit de parada (P).

Para la transmisión de datos se genera un pulso de reloj por cada bit de datos transferido.

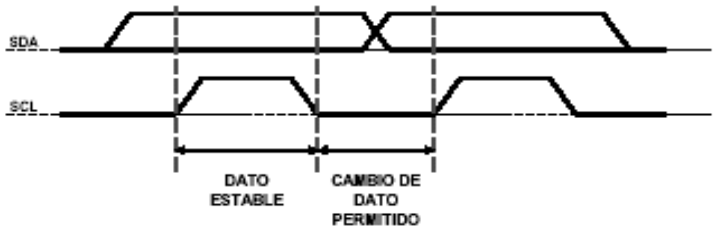


Figura 1.8 Transmisión de datos[24]

Los datos en la línea SDA deben permanecer estables, cuando las líneas de reloj, SCL está a nivel alto. Los datos en SDA solo pueden cambiar cuando SCL está a nivel bajo.

El bit de ACK se produce con el noveno pulso de reloj. Se produce cuando el esclavo (o el maestro si está recibiendo datos del esclavo) fuerza a nivel bajo la línea de datos, que previamente (al término del octavo pulso del reloj) se había puesto a nivel alto (por el maestro o el esclavo) en el momento en que la línea de reloj está a nivel alto. En la figura 9 se representa la formación del bit de acuse de recepción.

[24] http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf

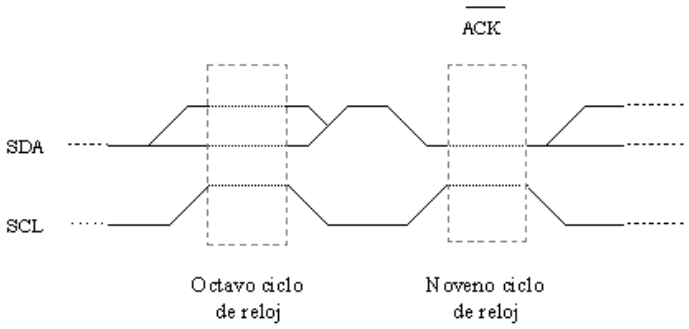


Figura 1.9 Formación del Bit de acuse de recepción[25]

1.3.3 MEMORIAS SERIALES 24LC08/24LC64[26]

Estas memorias son de tipo EEPROM y como principal cualidad permite el almacenamiento y sobre-escritura de datos. Estas memorias sostienen la información por muchos años sin fuente de alimentación.

Con las memorias EEPROM de interfaz serial, el control se ha reducido solamente a unos cuantos pins que son utilizados para entrada o salida de datos en forma serial, sincronismo, direccionamiento y por último los pins de alimentación del circuito.

La velocidad de transferencia de datos puede variar desde los 100 KHz hasta los 600 MHz, dependiendo del tipo de memoria y del sistema de comunicación utilizado.

Un aspecto que podría significar una limitante para las memorias seriales es la velocidad de lectura, si se comparan con la EEPROM paralelas, aunque las velocidades que se logran son aceptables para la mayoría de aplicaciones.

[25] http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf
[26] Data Sheets 24AA08/24LC08B (http://ww1.microchip.com/downloads/en/devicedoc/21710c.pdf)
Data Sheets 24AA64/24LC64(http://ww1.microchip.com/downloads/en/DeviceDoc/21189f.pdf)

1.3.3.1 Características

Las memorias 24LC08 pueden almacenar hasta 1 Kbyte y está organizada en cuatro bloques, cada uno de 256*8 bits con interface serial alámbricas. Mientras que las memorias 24LC64 pueden almacenar hasta 8 Kbyte, se componen de 8 bloques cada uno con capacidad de 1 Kbyte

Tiene 2 buses de interfaz serial compatibles con I^2C, protección de escritura por hardware, puede ser operado como una ROM serial, retención de información mayor a 200 años y tiene 8 pins.

Este dispositivo trabaja como un esclavo. Su diseño de bajo voltaje permite operar bajo los 1.8 V con una corriente de Standby y de operatividad de solamente 1µA y 1mA respectivamente. La frecuencia del reloj máxima es de 400 KHz ($Vcc \geq 2.5V$) y para $Vcc < 2.5V$ se tiene frecuencias de 100 KHz.

Las 24LC08 disponen de un sistema de protección de ruido, para esto emplean un detector umbral de Vcc que deshabilita la lógica interna de borrado/escritura si el voltaje está bajo los 1.5V en la condición nominal.

Las entradas SCL y SDA tienen un SCHMITT TRIGGER y un circuito filtro que suprime los picos de ruido para asegurar una apropiada operación del dispositivo, aún sobe ruidos en el bus.

En el presente proyecto se utiliza la memoria 24LC08B porque la capacidad que tienen es suficiente para los datos que se almacenan. En la sección siguiente se realiza un análisis más detallado de estas memorias.

1.3.3.2 Direccionamiento

El primer byte que se recibe del maestro después de la condición de inicio es el Byte de control, figura 1.10

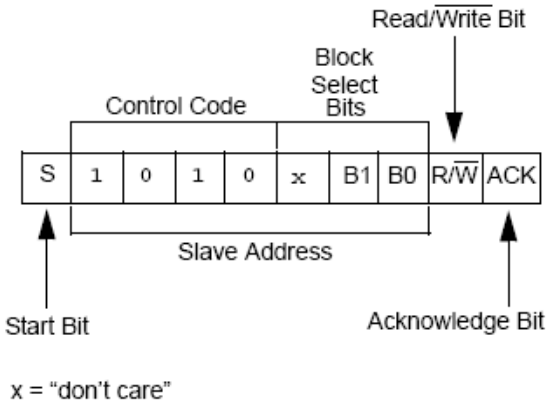


Figura 1.10 Byte de control[27]

Este byte consiste de

- Un código de control de 4 bits, para el 24LC08B es puesto en 10101 para operaciones de lectura y escritura.
- 3 bits para la selección del bloque (B2, B1, B0). El bloque B2 no se utiliza para el 24LC08B. Estos bits son usados por el maestro para seleccionar uno de los 4 bloques de memoria que será accedidos, estos bits son los más importantes del byte de control.
- El último bit define la operación que se realizará (Lectura o escritura)

$$R/\overline{W} = 1 \rightarrow \text{Operación de lectura}$$

$$R/\overline{W} = 0 \rightarrow \text{Operación de escritura}$$

[27] Data Sheets 24AA08/24LC08B

1.3.3.3 Escritura/Lectura

Escritura.- Cuando el bit $R/\overline{W}$ recibido es 0, el esclavo sabe que un byte con una dirección de palabra seguirá una vez que se ha generado un bit ACK durante el noveno ciclo de reloj. Por lo tanto, el próximo byte transmitido por el maestro es la "dirección de palabra" y será escrito dentro del puntero de dirección del dispositivo 24LC08B.

Después de recibir otra señal ACK, el maestro transmite el dato que será escrito dentro de la localidad de memoria seleccionado.

Otra vez el 24LC08B genera el ACK y el maestro genera la condición de parada, esto inicializa el ciclo de escritura interna y durante este tiempo la memoria no generará la señal de ACK.

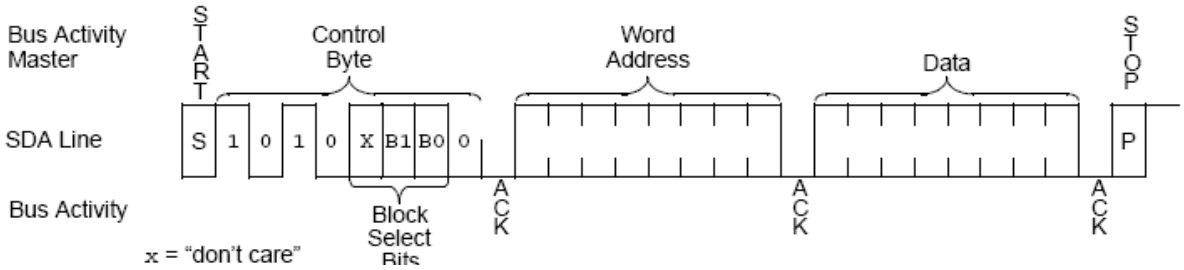


Figura 1.11 Escritura[28]

Lectura.- la operación de lectura se inicializa en la misma forma que la operación de escritura, con la excepción de que $R/\overline{W}$ es 1. Hay tres tipos básicos de operación de lectura:

- Lectura de direcciones actuales
- Lectura aleatoria

[28] Data Sheets 24AA08/24LC08B

- Lectura secuencial

Lectura de direcciones actuales: el 24LC08B contiene una "dirección de contador" que mantiene la dirección de la última palabra accedida, internamente incrementada en 1. Por lo tanto, si el acceso previo (escritura o lectura) fue en la dirección n, la próxima dirección en curso de la operación de lectura deberá accederse a los datos de la dirección n+1. Si el 24LC08B recibe su dirección con $R/\overline{W}$ en 1, genera un ACK y transmite el dato de 8 bits. El maestro transferirá un NO ACK pero genera la condición de parada y el 24LC08B discontinúa la transmisión.

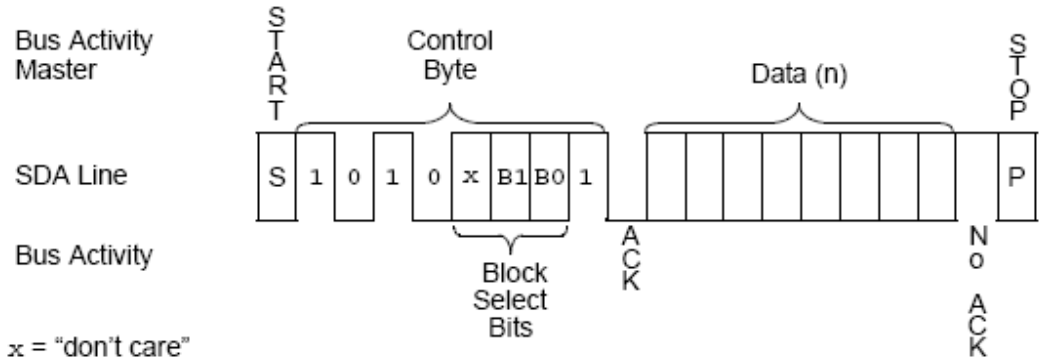


Figura 1.12 Lectura de direcciones actuales[29]

Lectura aleatoria: permite al maestro acceder a cualquier localidad de la memoria de forma aleatoria. Al ejecutar este tipo de lectura, primero se debe poner la dirección del dato. Esto se realiza enviando la dirección de la palabra al 24LC08B como parte de la operación de escritura. Una vez que la dirección de la palabra es enviada, el maestro genera la condición de inicio después del ACK y genera otra vez el byte de control pero con $R/\overline{W}$ en 1. La memoria genera un ACK y transmite la

[29] Data Sheets 24AA08/24LC08B

palabra de 8 bits, el maestro transferirá un NO ACK pero genera la condición STOP y el 24LC08B discontinúa la transmisión.

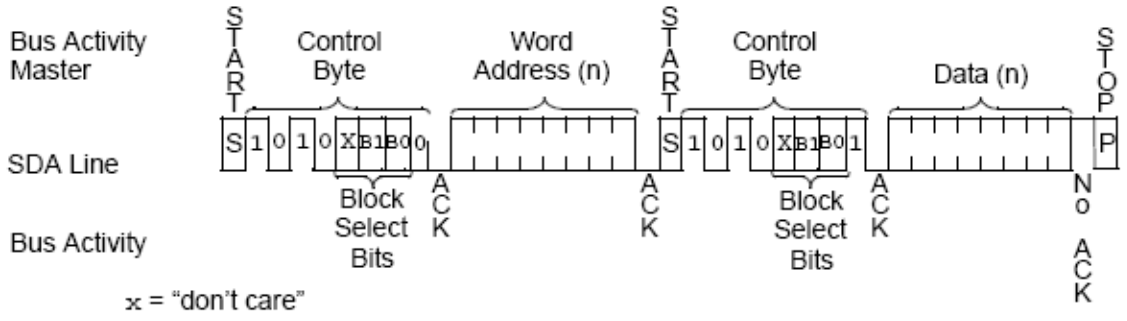


Figura 1.13 Lectura Aleatoria

Lectura Secuencial: se inicia de la misma forma que la lectura aleatoria, excepto que una vez que transmite el 24LC08B el primer byte de datos, el maestro emite un ACK como oponiéndose a la condición de parada y la memoria en forma directa transmite el próximo dato de 8 bits.

Para proveer la lectura secuencial, la memoria contiene un puntero de dirección interna que es incrementado en uno cuando se completa cada operación. Este puntero de dirección permite al contenido de la memoria entera ser leída serialmente durante una operación.

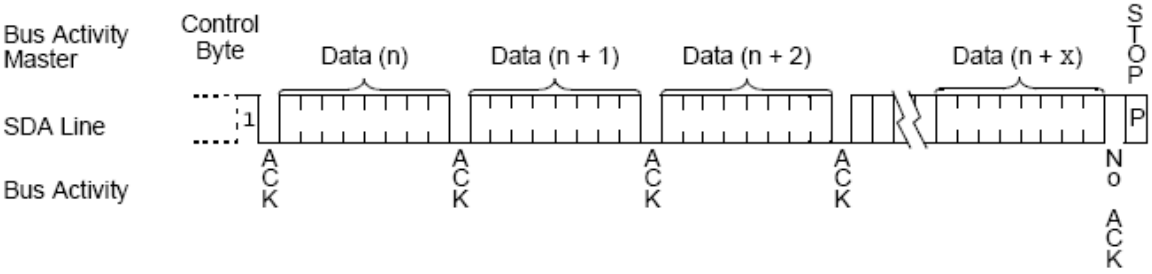


Figura 1.14 Lectura secuencial[30]

[30] Data Sheets 24AA08/24LC08B

1.3.3.4 Descripción de pins

Las memorias 24LC08B disponen de 8 pins, la distribución y descripción de cada uno de ellos se observa en la figura 1.15 y en la tabla 1.8

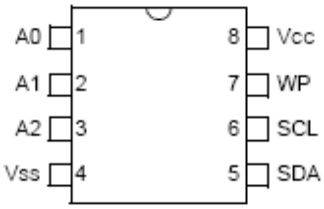


Figura 1.15 Distribución de pins

PIN	NOMBRE	DESCRIPCIÓN
1	A0	Sin conexión
2	A1	Sin conexión
3	A2	Sin conexión
4	Vss	Tierra
5	SDA	Entrada/Salida de datos
6	SCL	Señal de reloj
7	WP	Protección de escritura
8	Vcc	Voltaje de entrada

Tabla 1.8 Descripción de pins 24LC08B

El SDA es un pin bidireccional usado para transferir direcciones y datos dentro y fuera del dispositivo. Ya que este es un terminal de colector-abierto requiere resistencias de pull-up hacia Vcc, típicamente de 10 KΩ para 100 KHz y 2 KΩ para 400 KHz.

La entrada SCL es usado para sincronizar la transferencia de datos hacia y desde el dispositivo.

El pin WP se puede conectar a la señal de Tierra o Vcc. Si va hacia Vcc, las operaciones de escritura son deshabilitadas. La memoria entera será protegida contra escritura. Las operaciones de lectura no son afectadas.

Los pins A0, A1 y A2 no son usados por el 24LC08B, estos pins pueden quedar como flotantes o conectarse a Vss o Vcc.

1.4 MÓDULOS DE TRANSMISIÓN INALÁMBRICA

1.4.1 CONSIDERACIÓN SOBRE ANTENAS[31]

Una antena es un sistema conductor metálico capaz de radiar y capturar ondas electromagnéticas. Las antenas se usan para conectar las líneas de transmisión al espacio libre, el espacio libre a las líneas de transmisión, o ambas cosas. En resumen, una línea de transmisión acopla la energía de un transmisor o de un receptor a una antena, que a su vez acopla la energía con la atmósfera terrestre, y de la atmósfera terrestre a la línea de transmisión.

En un sistema de comunicación inalámbrica, la selección del tipo de antena es una consideración crítica. El rango y desempeño de un enlace de RF depende mucho de este requerimiento.

Algunos módulos de transmisión (por ejemplo los módulos Linx) tienen, comúnmente, una potencia de salida un poco mayor al límite establecido. Esto permite el uso de una antena no tan eficiente, como antenas tipo lazo o helicoidales, permitiendo cumplir mejor con los requerimientos de costo, dimensiones o apariencia y, aun así, conseguir la mayor potencia permitida a un rango máximo. Si se usa una antena con mejor eficiencia, es posible el ajuste de la potencia de salida.

[31] TOMASI, Wayne. “Sistemas de Comunicaciones Electrónicas”. Editorial Prentice - Hall. 2003

1.4.1.1 Atenuador tipo T[32]

Aunque hay numerosos métodos de obtener una reducción en la potencia de salida, uno de los más simples es un atenuador T formado por tres resistencias.

El propósito del atenuador es proveer una atenuación fija mientras se mantiene un acoplamiento de impedancias en 50 ohmios entre la antena y el transmisor. La figura 1.16 muestra un atenuador tipo T.

En esta configuración, las impedancias de 50 ohmios de la fuente (transmisor) y la carga (antena) forman divisores de voltaje que reducen el nivel en la potencia de salida. La selección de las resistencias permite que la entrada y salida del atenuador estén acopladas a las impedancias de la fuente y la carga para prevenir la atenuación indeseable debido a un desacoplamiento.

Los valores de resistencia se pueden calcular fácilmente cuando se conocen la atenuación deseada y las impedancias de entrada y salida. La tabla 1.9 permite determinar los valores de R1 y R2, está normalizada a impedancias de entrada y salida de 1 ohmio y asume que el atenuador está acoplando impedancias iguales en la fuente y en la carga.

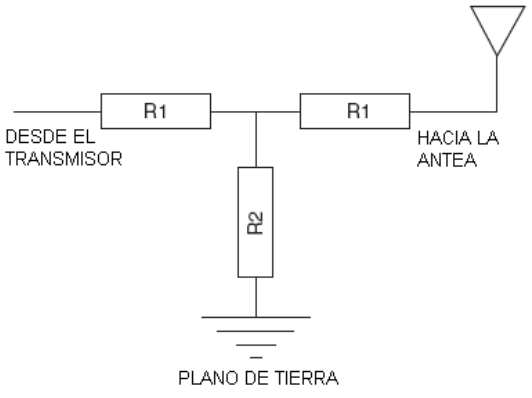


Figura 1.16 Atenuador Tipo T[33]

[32] http://www.linxtechnologies.com/Documents/AN-00150.pdf

Atenuación [dB]	R1	R2	Atenuación [dB]	R1	R2
1	0.0575	8.668	13	0.6342	0.4712
2	0.1147	4.305	14	0.6673	0.4155
3	0.1708	2.838	15	0.6980	0.3668
4	0.2263	2.097	16	0.7264	0.3238
5	0.2800	1.645	18	0.7764	0.2559
6	0.3323	1.339	20	0.8182	0.2020
7	0.3823	1.117	25	0.8935	0.1127
8	0.4305	0.9458	30	0.9387	0.0633
9	0.4762	0.8118	35	0.9650	0.0356
10	0.5195	0.7032	40	0.9818	0.0200
11	0.5605	0.6120	45	0.9888	0.1120
12	0.5985	0.5362	50	0.9937	0.00633

Tabla 1.9 Cálculo de R1 y R2[34]

La antena de recepción se optimiza para la frecuencia en la que el receptor opera, tratando de minimizar la recepción de señales no deseadas.

Entre los tipos de antena más comunes usados en aplicaciones de baja potencia, están las antenas tipo hélice o helicoidales, tipo lazo o de cuadro y tipo látigo, estas últimas también conocidas como antenas omnidireccionales o monopolos.

1.4.1.2 Atenuador tipo hélice o helicoidales[35]

[33] http://www.linxtechnologies.com/Documents/AN-00150.pdf
[34] http://www.linxtechnologies.com
[35] http://www.linxtechnologies.com/Documents/AN-00500.pdf

Una antena helicoidal es un cable enrollado usualmente de acero, cobre o latón formando espirales. Los elementos helicoidales reducen significativamente el tamaño físico de la antena pero presenta un reducido ancho de banda y el espaciamiento entre las espiras afecta mucho a su rendimiento. Además, una antena helicoidal es propensa a un rápido desajuste (detunning) especialmente por la proximidad de objetos. Por lo que es recomendable utilizar antenas prefabricadas, optimizadas para obtener un máximo desempeño.

En la figura 1.17 se pueden apreciar modelos de antenas helicoidales, encapsuladas o expuestas para montaje interno.

Figura 1.17 Modelo de Antena Helicoidal[36]

La antena helicoidal es una antena de VHF o UHF, ideal para aplicaciones donde se requiere irradiar ondas electromagnéticas de polarización circular, más que polarización horizontal o vertical.

Una antena helicoidal típica tiene desde un mínimo de 3 o 4 hasta un máximo de cerca de 20 vueltas, y ganancias de potencia de 15 a 20 dB. Las antenas helicoidales producen anchos de banda desde ±20 % de la frecuencia central, hasta un intervalo de 2:1 entre las frecuencias máxima y mínima de operación.

[36] http://www.linxtechnologies.com/Documents/AN-00500.pdf

1.4.1.3 Atenuador tipo lazo o de cuadro[37]

Una antena de cuadro o antena de lazo está formada por una o más espiras conductoras formando un cuadrado, hexágono, octógono o círculo, con dimensiones físicas que pueden variar según la frecuencia y el rendimiento esperado. La inductancia de la o las espiras habitualmente se sintoniza con un capacitor variable y se acopla a la línea de transmisión mediante un eslabón o mediante un acoplamiento capacitivo.

La figura 1.18 muestra una antena tipo lazo mostrando los elementos que la constituyen.

Este tipo de antenas también pueden ser incluidas en la placa de circuitos impresos (Printed Circuit Board, PCB) mediante líneas de transmisión de cinta y de microcinta, usando las pistas (o trazas) de cobre sobre la misma tarjeta de circuito impreso, lo que implica un muy bajo costo en su implementación.

A pesar de sus ventajas en relaciona a costos, estas antenas, llamadas antenas PCB, son generalmente ineficientes. Una antena tipo lazo puede ser muy difícil de ajustar y acoplar, siendo además muy sensitiva a los cambios en la constante del dieléctrico.

Las antenas fabricadas mediante líneas de transmisión de cinta o de microcinta, incluso se comercializan como “chips” que se montan directamente en cualquier circuito impreso. La figura 1.19 muestra diferentes modelos de antenas tipo lazo.

[37] http://www.linxtechnologies.com/Documents/AN-00500.pdf

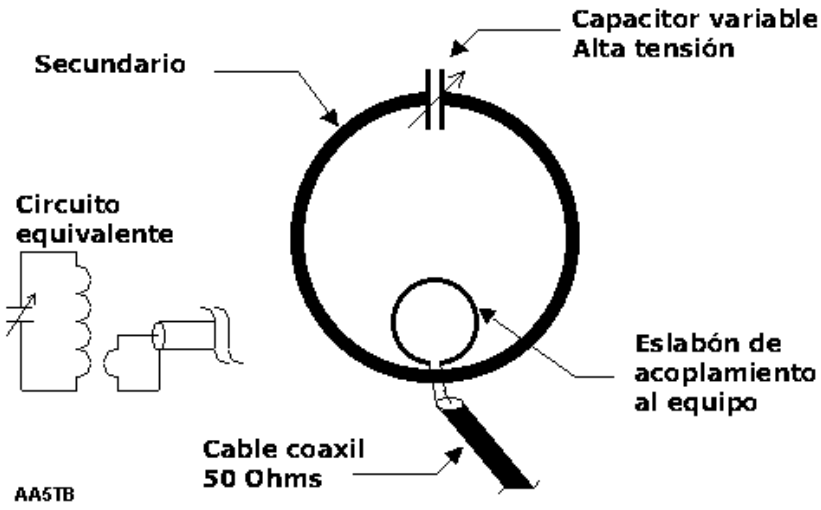


Figura 1.18 Elementos de la antena Tipo Lazo[38]

Figura 1.19 Modelos de Antena tipo Lazo[39]

1.4.1.4 Antenas tipo látigo (Omnidireccionales o monopolos)[40]

Este tipo de antenas llamada monopolo o antena de Marconi tiene generalmente dimensiones de media longitud de onda o un cuarto de longitud de onda.

[38] http://www.solred.com.ar/lu6etj/tecnicos/loop/antena_de_cuadro.htm
[39] http://www.linxtechnologies.com/Documents/AN-00500.pdf
[40] http://www.linxtechnologies.com/Documents/AN-00500.pdf

Presenta características similares a la antena de Hertz o dipolo cuando está conectada directamente al suelo o a un plano de referencial de tierra.

Una antena tipo látigo provee mucha estabilidad y buen desempeño. Posee una amplio ancho de banda y puede ser diseñada e integrada con relativa facilidad a la mayoría de diseños. Es posible optar por antenas de construcción profesional, prefabricadas y que estéticamente ofrecen una mejor apariencia. Estas antenas generalmente están formadas por un conductor encapsulado en un cubierta plástica o de goma.

Muchas veces, cuando la longitud de la antena excede a los requerimientos deseados, ésta se combina con un doblez tipo hélice para reducir la dimensión total de la antena. Esta técnica se conoce como ensanchamiento de base ("base loading"). La figura 1.20 muestra algunos modelos de antenas tipo látigo.

Figura 1.20 Antenas Tipo Látigo[41]

En la tabla 1.10 se comparan varias de las características principales de las antenas antes mencionadas.

[41] http://www.linxtechnologies.com/Documents/AN-00500.pdf

Parámetro	De lazo	Helicoidal	De látigo
Desempeño general	●	●●	●●●
Facilidad de instalación	●	●●	●●●
Tamaño	●●	●●●	●●
Inmunidad a efectos de proximidad	●●●	●●	●
Rango en campo abierto.	100 pies	200 pies	300+ pies
● =Regular ●● Bueno ●●● =Excelente			

Tabla 1.10 Características de varios tipos de Antena[42]

De las antenas mencionadas se escoge el tipo Látigo, por su facilidad de instalación, ya que un solo alambre puede hacer la función de antena.

1.4.2 MÓDULOS INALÁMBRICOS

1.4.2.1 Módulos LINX serie LC[43]

La serie LC está idealmente concebida para su uso masivo en aplicaciones como control a distancia, seguridad, identificación y transmisión periódica de datos. Empacado en un compacto formato SMD. Tienen un diseño SAW (Surface Acoustic Wave), Con excepción de una antena, no se requieren componentes RF externos, Este tipo de modulación AM es conocido también como CW (Continuos Wave) y

[42] http://www.linxtechnologies.com/
[43] http://www.linxtechnologies.com/Documents/AN-00130.pdf

OOK (On-Off Keying), representa un estado lógico bajo “0” como la ausencia de portadora y un estado lógico alto “1” por la presencia de la portadora(figura 1.21).

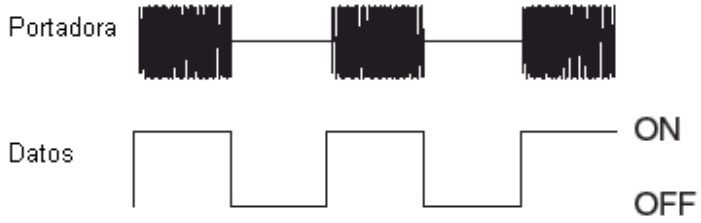
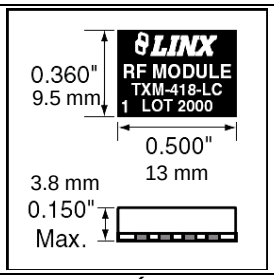


Figura 1.21 Modulación CPCA

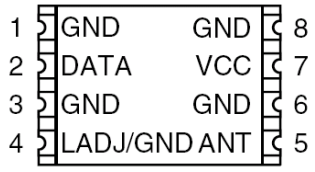
Módulo transmisor

Cuando se lo utiliza con un Receptor compatible de la serie LC, se forma un enlace altamente confiable, capaz de transferir datos a distancias de hasta 300 pies (aprox. 100 m).

En la figura 1.22 se presenta al módulo con sus dimensiones físicas y con la distribución de pins



DIMENSIONES FÍSICAS



CONFIGURACIÓN DE PINS

Figura 1.22 Módulo Transmisor LINX[44]

[44] http://www.linxtechnologies.com/

Este transmisor tiene modulación CPCA (Carrier-Present Carrier-Absent) basado en un oscilador SAW, Posee un alto desempeño a bajo costo siendo capaz de transmitir datos hasta 5000 bps. La figura 1.23 muestra el diagrama de bloques de un transmisor de la serie LC de Linx.

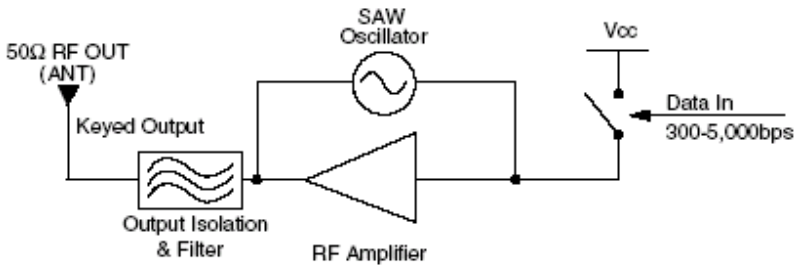
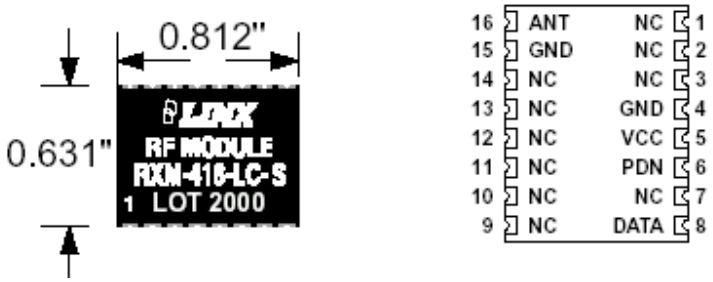


Figura 1.23 Diagrama de bloques de un Transmisor serie LC[45]

Modulo receptor

En la figura 1.24 se muestran al módulo receptor, la distribución de pins y la forma de conectar a este módulo, para recepción de datos.



[45] http://www.linxtechnologies.com/Documents/TXM-xxx-LC_Data_Guide.pdf

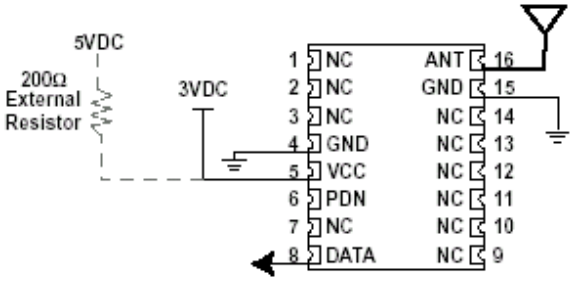


Figura 1.24 Módulo Receptor LINX

El Receptor recoge datos enviados por un CPCA, y la convierte en datos binarios.

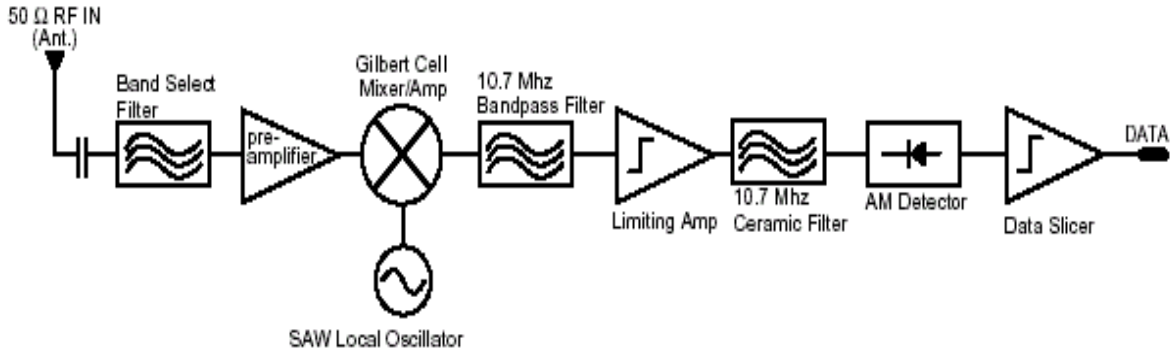


Figura 1.25 Diagrama de bloques Receptor LC[46]

1.4.2.2 Módulos TLP434/RLP434[47]

Estos módulos son fabricados por Laipac. Ofrece varios modelos de radiofrecuencia de bajo coste para transmisión de datos a gran velocidad incluyendo las series TLP/RLP de transmisores y receptores. Estos módulos proveen un simple par de transmisor/receptor para proyectos inalámbricos de bajo costo.

[46] http://www.linxtechnologies.com/Documents/RXM-xxx-LC-S_Data_Guide.pdf
[47] http://www.laipac.com/easy_434_esp.htm

Los transmisores tienen un rango típico de alcance de unos 150m en áreas abiertas, tienen modulación ASK con una salida de 8mW dependiendo del voltaje con que se alimente al módulo. La sensibilidad típica de los receptores está en los 3 µV. En la figura 1.26 se muestra un par de estos módulos.

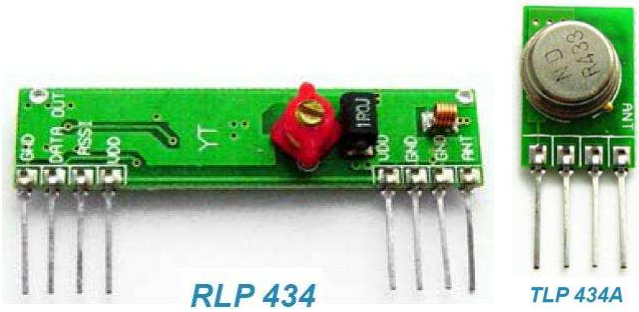


Figura 1.26 Par Tx/Rx Fabricante Laipac[48]

Mientras se establece la comunicación entre dos circuitos separados, hay dos consideraciones que ampliarán el rango y la eficiencia de estos dispositivos.

Ante todo cada dispositivo puede transmitir desde 14dbm a 16dbm de potencia, pero esto es cuando el módulo es alimentado con 12 V. Comúnmente el Microcontrolador y otros dispositivos funcionan a 5 voltios, lo que se necesitaría una línea de poder separado.

También el uso de una antena es opcional. Si se la usa, esta incrementará la eficiencia de la comunicación inalámbrica. La antena más larga funciona mejor.

El fabricante recomienda entre 30 y 35 cm de largo. También se lo puede hacer en un circuito impreso.

La figura 1.27 representa las pruebas hechas por el fabricante cuando se alimenta el Transmisor con diferentes voltajes.

[48] http://www.laipac.com/Downloads/Easy/tlp434a.pdf
http://www.laipac.com/Downloads/Easy/rlp434.pdf

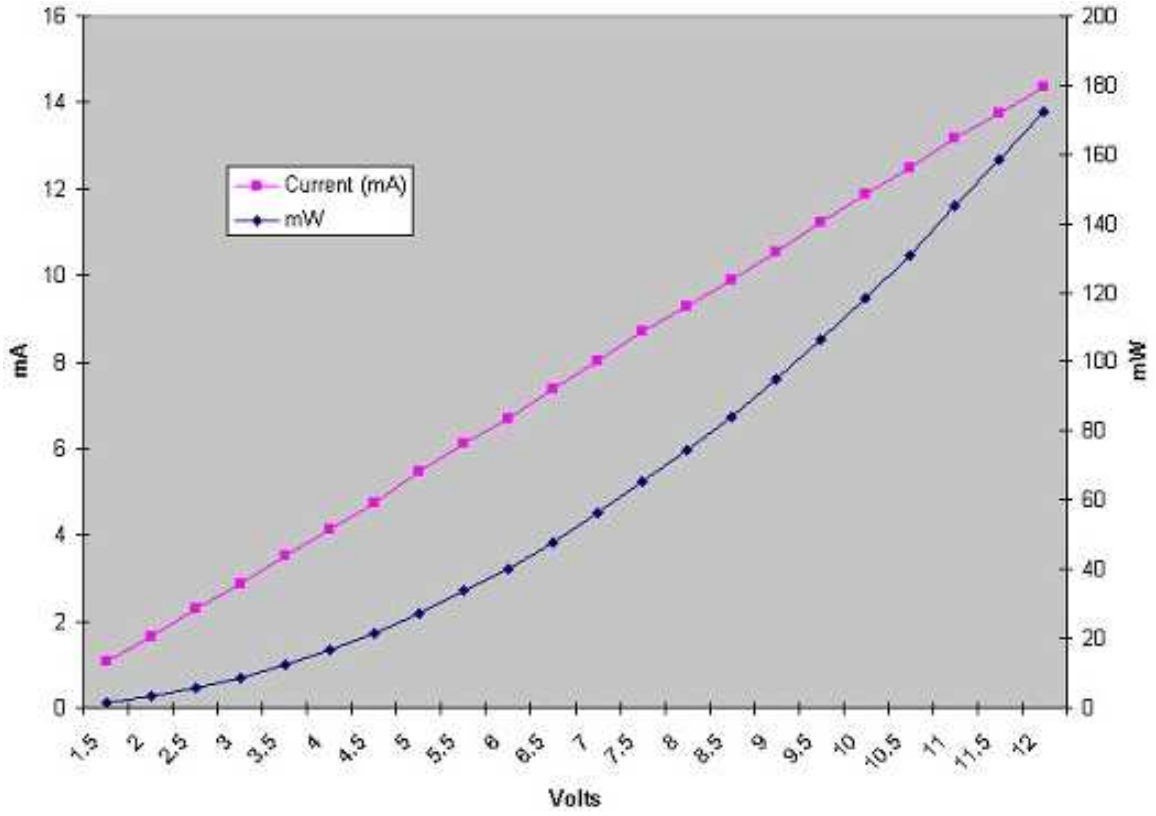


Figura 1.27 Voltaje vs Potencia[49]

Según las pruebas que hace el fabricante (Laipac Tech), se pueden alcanzar distancias desde 60 a 130 pies (desde 18 a 40 m) inclusive en ambientes ruidosos, como el de RF ocasionado por motores AC

La tabla 1.11 muestra los diferentes modelos de Transmisores. Y en la tabla 1.12 se tiene los modelos receptores.

[49] http://www.laipac.com/Downloads/Easy/tlp_power.pdf

MÓDULOS TRANSMISORES					
MODELO	MODULACION	FRECUENCIA MHz	VOLTAJE V	VELOCIDAD Kbps	POTENCIA
TLP 315A	ASK	315	3 a 12	8	+ 8.8 dBm
TLP 418A	ASK	418	3 a 12	8	+ 8.8 dBm
TLP 434A	ASK	433.92	3 a 12	8	+ 8.8 dBm
TLP 868A	ASK	868.35	3 a 12	8	+ 8.8 dBm
TLP 868F	FSK	868.36	2.7 a 5.5	4 a 40	VARIABLE

Tabla 1.11 Módulos Transmisores[50]

MÓDULOS RECEPTORES					
MODELO	MODULACION	FRECUENCIA MHz	VOLTAJE V	VELOCIDAD Kbps	SENSIBILIDAD dBm
RLP 315	ASK	315	4.5 - 5.5	4.8	-116
RLP 315A	ASK	315	3.5 - 6	4.8	-116
RLP 418	ASK	418	5	4.8	-106
RLP 418A	ASK	418	3.5 - 6	4.8	-116
RLP 434	ASK	433.92	4.5 - 5.5	4.8	-116
RLP 434A	ASK	433.93	3.5 - 6	4.8	-116
RLP 868A	ASK	868.35 300 – 1000	5 2.7 - 5.5	3 - 100	-112
RLP 916A	ASK	916.5 300 – 1000	5 2.7 - 5.5	33 - 100	-112
RLP 868F	FSK	868.35 300 – 1000	2.7 - 5.5	3 - 100	-112
RLP16F	FSK	916.5	2.7 - 5.5	33 - 100	-112

Tabla 1.12 Módulos Receptores[51]

[50] http://www.laipac.com/

1.4.2.3 Módulos E-MadeinCHN[52]

Estos módulos son fabricados por la compañía E-MadeinCHN, tienen modulación ASK, basada en un oscilador SAW (Surface Acoustic Wave). En las tablas 1.13 y 1.14 se muestran algunos de los modelos de estos módulos.

MÓDULOS TRANSMISORES					
MODELO	VOLTAJE V	CORRIENTE mA	VELOCIDAD Kbps	ALCANCE M	FRECUENCIA MHZ
TM50-1	3 a 12	5 a 45	9.6	50 - 100	315, 433
TM1000-1	3 a 12	5 a 45	2.4	1000	315, 433
TM4000-1	9	> 100	2.4	4000	315, 433
TM100-4S	3 a 12	5 a 45		100	315, 433
TM1000-4S	3 a 12	5 a 45	2.4 (max 9.6)	1000	315, 433
TM1000-4	3 a 12	5 a 45	2.4 (max 9.6)	1000	315, 433
TM4000-4	9	> 100	2.4 (max 9.6)	4000	315

Tabla 1.13 Módulos Transmisores

MÓDULO RECEPTOR					
MODELO	VOLTAJE V	CORRIENTE µA	SENSIBILIDAD dBm	VELOCIDAD Kbps	FRECUENCIA MHz
RM1SGS	2.6 - 4.5	200	-93	2.4	315, 433
RM1SGS2	2.6 - 4.5	200	-93	2.4	315, 433
RM1SG	5	5000	-103	4.8	315, 433
RM4SG-L	5	5000	-103	4.8	315, 433

Tabla 1.14 Módulos Receptores

[51] http://www.laipac.com/
[52] http://www.e-madeinchn.com/TransimittingModules.html
http://www.e-madeinchn.com/ReceiverModules.html

Los módulos que se van a utilizar en el presente proyecto para transmitir/recibir son los modelos TM1000-1S/ RM1SG respectivamente, ya que son los más fáciles de encontrar en el mercado.

1.5 PARÁMETROS DE DISEÑO

1.5.1 MICROCONTROLADORES

Actualmente en el mercado existe una gran variedad de microcontroladores, para el presente proyecto se escogieron los modelos fabricados por la compañía Microchip.

1.5.1.1 PIC18F4550-I/PT

El DIFIJO se conecta a una PC para visualizar los datos del DIMOVIL, y se lo hace a través de una conexión USB, por tal motivo se escogió el modelo PIC18F4550-I/PT, ya que dispone de un terminal USB.

En la figura 1.28 se muestra la distribución de pins, la tabla 1.15 enumera las características y la tabla 1.16 la función de cada pin.

CARACTERISTICAS	PIC18F4550
Frecuencia de operación	Hasta 48MHz
Memoria de Programa (bytes)	32768
Memoria RAM de datos (bytes)	2048
Memoria EEPROM Datos (bytes)	256
Interrupciones	20
Líneas de E/S	35
Temporizadores	4
Módulos de comparación/Captura/PWM (CCP)	1

Módulos de Comparación/Captura/PWM mejorado (ECCP)	1
Canales de comunicación serie	MSSP, EUSART
Canal USB	1
Puerto Paralelo de Transmisión de Datos (SPP)	1
Canales de Conversión A/D de 10 bits	1
Comparadores analógicos	2
Juego de Instrucciones	75

Tabla 1.15 Características PIC18F4550[53]

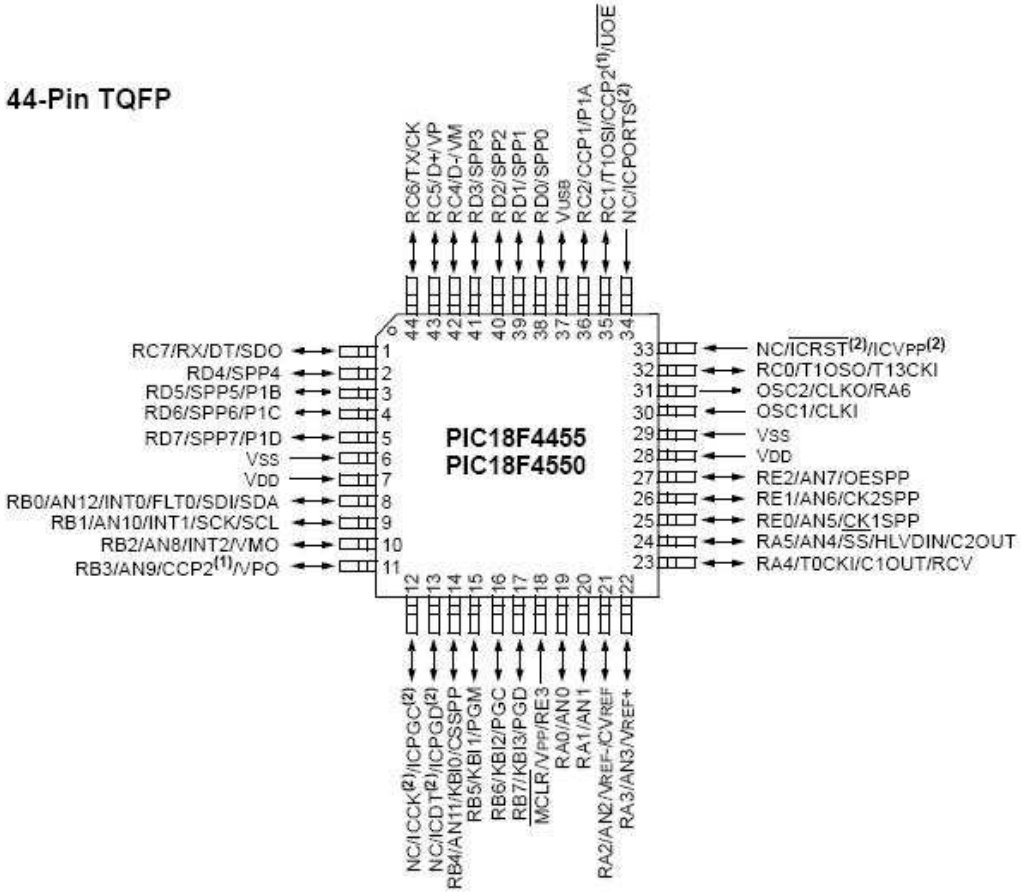


Figura 1.28 PIC18F4550-I/PT[54]

[53] http://www.i-micro.com/pdf/articulos/usb_imicro.pdf
[54] http://ww1.microchip.com/downloads/en/DeviceDoc/39632b.pdf

NÚMERO DE PIN	NOMBRE DEL PIN	DESCRIPCIÓN
1	RC7/RX/DT/SDO	RC7: Digital I/O RX: EUSART asynchronous receive DT: EUSART synchronous data SDO: SPI data out
2	RD4/SPP4	RD4: Digital I/O SPP4: Streaming Parallel Port data
3	RD5/SPP5/P1B	RD5: Digital I/O SPP5: Streaming Parallel Port data P1B: Enhanced CCP1 PWM output, channel B
4	RD6/SPP6/P1C	RD6: Digital I/O SPP6: Streaming Parallel Port data P1C: Enhanced CCP1 PWM output, channel C
5	RD7/SPP7/P1D	RD7: Digital I/O SPP7: Streaming Parallel Port data P1D: Enhanced CCP1 PWM output, channel D
6	V_{SS}	Ground reference for logic and I/O pins
7	V_{DD}	Positive supply for logic and I/O pins
8	RB0/AN12/INT0/FLT0/SDI/SDA	RB0: Digital I/O AN12: Analog input 12 INT0: External interrupt 0 FLT0: Enhanced PWM Fault input (ECCP1 module) SDI: SPI data in SDA: I^2C™ data I/O
9	RB1/AN10/INT1/SCK/SCL	RB1: Digital I/O AN10: Analog input 10 INT1: External interrupt 1 SCK: Synchronous serial clock input/output for SPI mode SCL: Synchronous serial clock input/output for I^2C mode
10	RB2/AN8/INT2/VMO	RB2: Digital I/O AN8: Analog input 8 INT2: External interrupt 2 VMO: External USB transceiver VMO output
11	RB3/AN9/CCP2/VPO	RB3: Digital I/O AN9: Analog input 9 CCP2: Capture 2 input/Compare 2 output/PWM 2 output VPO: External USB transceiver VPO output
12	NC	NC

13	NC	NC
14	RB4/AN11/KBIO/CSSPP	RB4: Digital I/O AN11: Analog input 11 KBI0: Interrupt-on-change pin CSSPP: SPP chip select control output
15	RB5/KBI1/PGM	RB5: Digital I/O KBI1: Interrupt-on-change pin PGM: Low-Voltage ICSP™ Programming enable pin
16	RB6/KBI2/PGC	RB6: Digital I/O KBI2: Interrupt-on-change pin PGC: In-Circuit Debugger and ICSP programming clock pin
17	RB7/KBI3/PGD	RB7: Digital I/O KBI3: Interrupt-on-change pin PGD: In-Circuit Debugger and ICSP programming data pin
18	$\overline{MCLR}$ /V_{PP}/RE3	MCLR: Master Clear (Reset) input. This pin is an active-low V_{PP}: Programming voltage input RE3: Digital input
19	RA0/AN0	RA0: Digital I/O AN0: Analog input 0
20	RA1/AN1	RA1: Digital I/O AN1: Analog input 1
21	RA2/AN2/V_{REF}-/CV_{REF}	RA2: Digital I/O AN2: Analog input 2 V_{REF}-: A/D reference voltage (low) input CV_{REF}: Analog comparator reference output
22	RA3/AN3/V_{REF}+	RA3: Digital I/O AN3: Analog input 3 V_{REF}+: A/D reference voltage (high) input
23	RA4/T0CKI/C1OUT/RCV	RA4: Digital I/O T0CKI: Timer0 external clock input C1OUT: Comparator 1 output RCV: External USB transceiver RCV input
24	RA5/AN4/ $\overline{SS}$ /HLVDIN/C2OT	RA5: Digital I/O AN4: Analog input 4 SS: SPI slave select input HLVDIN: High/Low-Voltage Detect input C2OUT: Comparator 2 output

25	RE0/AN5/CK1SPP	RE0: Digital I/O AN5: Analog input 5 CK1SPP: SPP clock 1 output
26	RE1/AN6/CK2SPP	RE1: Digital I/O AN6: Analog input 6 CK2SPP: SPP clock 2 output
27	RE2/AN7/OESPP	RE2: Digital I/O AN7: Analog input 7 OESPP: SPP output enable output
28	V_{DD}	Positive supply for logic and I/O pins
29	V_{SS}	Ground reference for logic and I/O pins
30	OSC1/CLKI	OSC1: Oscillator crystal input or external clock source input CLKI: External clock source input. Always associated with pin function OSC1
31	OSC2/CLKO/RA6	OSC2: Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode CLKO: In RC mode, OSC2 pin outputs CLKO which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate RA6: General purpose I/O pin
32	RC0/T1OSO/T13CKI	RC0: Digital I/O T1OSO: Timer1 oscillator output T13CKI: Timer1/Timer3 external clock input
33	NC	NC
34	NC	NC
35	RC1/T1OSI/CCP2/$\overline{UOE}$	RC1: Digital I/O T1OSI: Timer1 oscillator input CCP2: Capture 2 input/Compare 2 output/PWM 2 output UOE: External USB transceiver OE output
36	RC2/CCP1/P1A	RC2: Digital I/O CCP1: Capture 1 input/Compare 1 output/PWM 1 output P1A: Enhanced CCP1 PWM output, channel A
37	V_{USB}	Internal USB 3.3V voltage regulator output
38	RD0/SPP0	RD0: Digital I/O SPP0: Streaming Parallel Port data
39	RD1/SPP1	RD1: Digital I/O SPP1: Streaming Parallel Port data
40	RD2/SPP2	RD2: Digital I/O SPP2: Streaming Parallel Port data

41	RD3/SPP3	RD3: Digital I/O SPP3: Streaming Parallel Port data
42	RC4/D-/VM	RC4: Digital input D-: USB differential minus line (input/output) VM: External USB transceiver VM input
43	RC5/D+/VP	RC5: Digital input D+: USB differential plus line (input/output) VP: External USB transceiver VP input
44	RC6/TX/CK	RC6: Digital I/O TX: EUSART asynchronous transmit CK: EUSART synchronous clock (see RX/DT)

Tabla 1.16 Distribución de Pins PIC18F4550-I/PT[55]

1.5.1.2 PIC16F876A-I/SO

En el DIMOVIL se puede utilizar muchos modelos de microcontroladores, para este proyecto se utiliza el PIC16F876A-I/SO, es de menor características que el utilizado en el DIFIJO pero suficiente para los requerimientos que se necesita en este dispositivo. La figura 1.29 muestra la distribución de pins, la tabla 1.17 las características de este microcontrolador y la tabla 1.18 la función de cada pin.

CARACTERÍSTICAS	PIC16F876A
Frecuencia de operación	Hasta 20MHz
Memoria de Programa (bytes)	8KB
Memoria RAM (bytes)	368
Memoria EEPROM (bytes)	256
Interrupciones	13
Líneas de E/S	A, B y C
Temporizadores	3
Módulos de comparación/Captura/PWM (CCP)	2
Canales de comunicación serie	MSSP, EUSART
Juego de Instrucciones	35
Pins	28

Tabla 1.17 Características PIC16F876A

[55] http://ww1.microchip.com/downloads/en/DeviceDoc/39632b.pdf

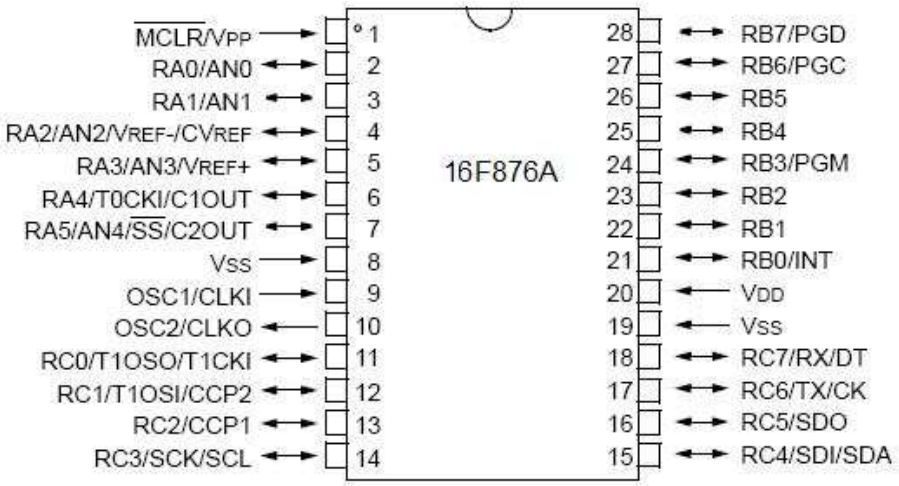


Figura 1.29 PIC16F876A-I/SO[56]

NÚMERO DE PIN	NOMBRE DEL PIN	DESCRIPCIÓN
1	$\overline{\text{MCLR}}$ /VPP	MCLR: Master Clear (input) or programming voltage (output). Master Clear (Reset) input. This pin is an active low Reset to the device VPP: Programming voltage input
2	RA0/AN0	RA0: Digital I/O AN0: Analog input 0
3	RA1/AN1	RA1: Digital I/O AN1: Analog input 1
4	RA2/AN2/VREF-/CVREF	RA2: Digital I/O AN2: Analog input 2 VREF-: A/D reference voltage (Low) input CVREF: Comparator VREF output
5	RA3/AN3/VREF+	RA3: Digital I/O AN3: Analog input 3 VREF+: A/D reference voltage (High) input
6	RA4/T0CKI/C1OUT	RA4: Digital I/O – Open-drain when configured as output T0CKI: Timer0 external clock input C1OUT: Comparator 1 output

[56] http://datasheets.net/pdf/microchip.com/39582b.pdf

7	RA5/AN4/$\overline{SS}$ /C2OUT	RA5: Digital I/O AN4: Analog input 4 SS: SPI slave select input C2OUT: Comparator 2 output
8	VSS	Ground reference for logic and I/O pins
9	OSC1/CLKI	OSC1: Oscillator crystal input or external clock source input CLKI: External clock source input
10	OSC2/CLKO	OSC2: Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode CLKO: In RC mode, OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate
11	RC0/T1OSO/T1CKI	RC0: Digital I/O T1OSO: Timer1 oscillator output T1CKI: Timer1 external clock input
12	RC1/T1OSI/CCP2	RC1: Digital I/O T1OSI: Timer1 oscillator input CCP2: Capture2 input, Compare2 output, PWM2 output
13	RC2/CCP1	RC2: Digital I/O CCP1: Capture1 input, Compare1 output, PWM1 output
14	RC3/SCK/SCL	RC3: Digital I/O SCK: Synchronous serial clock input/output for SPI mode SCL: Synchronous serial clock input/output for I2C mode
15	RC4/SDI/SDA	RC4: Digital I/O SDI: SPI data in SDA: I2C data I/O
16	RC5/SDO	RC5: Digital I/O SDO: SPI data out
17	RC6/TX/CK	RC6: Digital I/O TX: USART asynchronous transmit CK: USART1 synchronous clock
18	RC7/RX/DT	RC7: Digital I/O RX: USART asynchronous receive DT: USART synchronous data
19	VSS	Ground reference for logic and I/O pins
20	VDD	Positive supply for logic and I/O pins

21	RB0/INT	RB0: Digital I/O INT: External interrupt
22	RB1	Digital I/O
23	RB2	Digital I/O
24	RB3/PGM	RB3: Digital I/O PGM: Low-voltage (single-supply) ICSP programming enable pin
25	RB4	Digital I/O
26	RB5	Digital I/O
27	RB6/PGC	RB6: Digital I/O PGC: In-circuit debugger and ICSP programming clock
28	RB7/PGD	RB7: Digital I/O PGD: In-circuit debugger and ICSP programming data

Tabla 1.18 Distribución de Pins PIC16F876A-I/SO[57]

1.5.2 MÓDULOS INALÁMBRICOS

Para el cobro de peaje necesariamente se tiene que utilizar un canal inalámbrico en el cual se transmitan los datos entre el DIFIJO y el DIMOVIL. Las dos opciones disponibles son la utilización de transceptores por luz infrarroja o bien por radio frecuencia. La primera es utilizada por lo general en los mandos a distancia con que vienen equipados la mayoría de los aparatos domésticos y comerciales y la segunda es más frecuente verla en juguetes, garajes, etc.

La principal ventaja de los transceptores de radio frecuencia frente a los infrarrojos son el mayor alcance y el que no requiera línea de vista entre transmisor y receptor y estos son los que se utiliza en los circuitos del DIFIJO y DIMOVIL.

Hay varias empresas que fabrican módulos de RF de variadas características pero los que se pueden encontrar en el mercado con mayor facilidad son los fabricados

[57] http://datasheets.net/pdf/microchip.com/39582b.pdf

por LAIPAC y por E-MadeinCHN, siendo los modelos de este último los más económicos.

En la figura 1.30 se observa los modelos utilizados en los circuitos del DIFIJO y DIMOVIL.

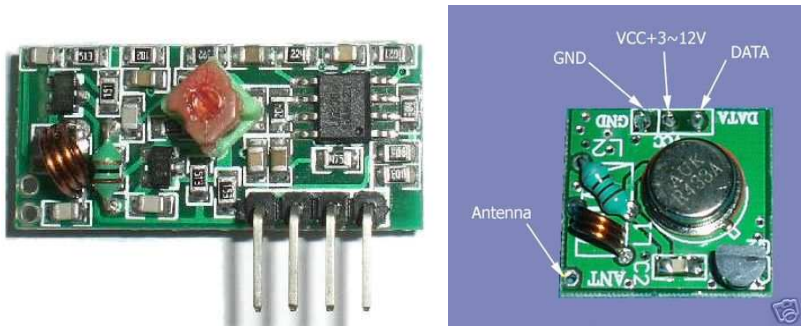


Figura 1.30 Par Transmisor Receptor E-MadeinCHN[58]

1.5.3 MEMORIA

En el dispositivo móvil se utiliza una memoria serial que tiene como función guardar el saldo que dispone, esta memoria se comunica con el microcontrolador mediante el protocolo I^2C. Se escogió la memoria 24LC08B (figura 1.31), La figura 1.32 muestra la distribución de pins de esta memoria y en tabla 1.19 se enumera la función de cada uno.

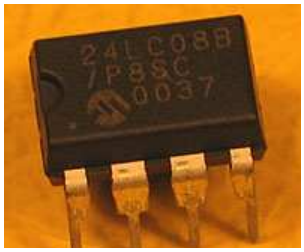


Figura 1.31 Memoria serial 24LC08B

[58] http://www.e-madeinchn.com/TransimittingModules.html
http://www.e-madeinchn.com/ReceiverModules.html

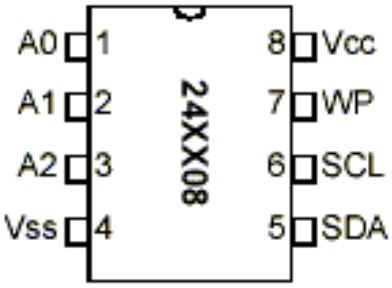


Figura 1.32 Memoria 24LC08_Pins[59]

PIN	NOMBRE	FUNCIÓN
1	A0	No se usan: Pueden quedar como flotantes o conectar a Vss o Vcc
2	A1	No se usan: Pueden quedar como flotantes o conectar a Vss o Vcc
3	A2	No se usan: Pueden quedar como flotantes o conectar a Vss o Vcc
4	Vss	Conexión a Tierra
5	SDA	Pin bidireccional para transferir direcciones y datos dentro y fuera del dispositivo
6	SCL	Usado para sincronizar la transferencia de datos hacia y desde el dispositivo
7	WP	Si se conecta a Vcc entonces funciona como ROM y a Vss habilita la función de escritura
8	Vcc	Fuente de energía (5V)

Tabla 1.19 Función Pins 24LC08B[60]

[59] http://ww1.microchip.com/downloads/en/devicedoc/21710c.pdf
[60] http://ww1.microchip.com/downloads/en/devicedoc/21710c.pdf

CAPÍTULO 2

2. DISEÑO Y CONSTRUCCIÓN DEL HARDWARE DEL PAGO AUTOMÁTICO DE PEAJE

2.1 INTRODUCCIÓN

En este capítulo se describe el desarrollo de prototipo de laboratorio para el cobro electrónico de peaje, tanto en hardware como en software, se define el protocolo de comunicaciones entre PC – DIFIJO y DIFIJO - DIMOVIL.

2.2 PROTOCOLO DE COMUNICACIONES

Cada DIMOVIL viene de fábrica grabado en la memoria del microcontrolador un único identificativo, que es la placa del auto y el tipo de vehículo: Muy Pesado, pesado, liviano y especial. El saldo que se recarga se graba en la memoria externa 24LC08B.

En la memoria del microcontrolador del DIFIJO viene grabado las tarifas de cobro para cada tipo de vehículo, un único identificativo, para este caso se tiene: “CASETA 1” y la forma de cobro (Manual o Automática), todos estos datos se puede modificar mediante la interfaz de usuario.

Cobro Manual.- Se lo denomina así porque cada vez que se acerca un auto que tenga incorporado el DIMOVIL, para realizar el cobro se presiona el botón “cobrar peaje” de la interfaz de usuario.

Cobro automático.- En este modo no es necesario presionar ninguna tecla para realizar el cobro, el DIFIJO automáticamente realiza el descuento del peaje, espera 7 segundos y nuevamente busca DIFIJOS para repetir el proceso. En el ANEXO F se detalla el manejo de dicha interfaz.

Las tramas que intervienen en la comunicación son de longitud variable y difieren una de otras en ciertos campos. En la figura 2.1 se observa el formato general de las tramas para la transmisión y recepción de datos a través de los módulos de radiofrecuencia.

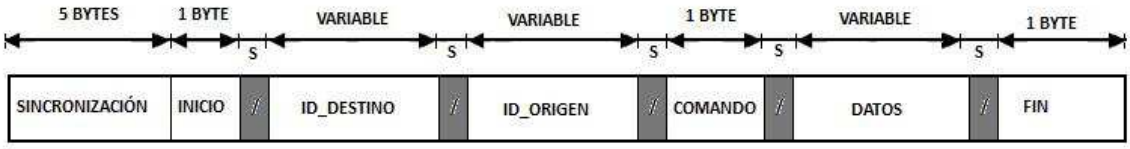


Figura 2.1 Formato General de una Trama

En la comunicación de RF antes de cada trama se envía 5 bytes que sirven para sincronizar a los dispositivos, de acuerdo a pruebas realizadas los bytes que mejor se sincronizan son:

BYTES DE SINCRONIZACIÓN				
AAH	AAH	50H	72H	20H

Tabla 2.1 Bytes de Sincronización

En el byte de inicio se envía el caracter asterisco "*". El ID_DESTINO es el identificativo del vehículo, es de longitud variable. El ID_ORIGEN es el identificativo del dispositivo que envía la trama, por ejemplo "CASETA 1".

En el segmento de datos se envían los valores que interesan a los dispositivos su contenido depende del tipo de trama. En el campo fin se envía el caracter "enter"

como byte de fin de trama. Entre el inicio de trama y el fin de trama cada bloque se encuentra separado por un byte (carácter “/”).

TRAMAS DE RF		
COMANDO (TIPO DE TRAMA)	BYTE (CARACTER)	DESCRIPCIÓN
SOLICITUD_TARIFAS PEAJE	1	PC → DIFIJO: El DIFIJO tiene grabado en su memoria las tarifas, para saber cuáles son y desplegar en la PC, enviamos esta trama.
RESPUESTA TARIFAS_PEAJE	2	DIFIJO → PC: El DIFIJO envía una trama a la PC con los datos de la tarifa.
SOLICITUD_ID	3	DIFIJO → DIMOVIL: Es la trama que envía el DIFIJO a todos los DIMOVIL (Broadcast).
RESPUESTA_ID	4	DIMOVIL → DIFIJO: Cuando un auto detecta la trama Broadcast, entonces envía una trama con la placa del auto.
SOLICITUD_SALDO	5	DIFIJO → DIMOVIL: Para saber que saldo tiene el vehículo.
RESPUESTA_SALDO	6	DIMOVIL → DIFIJO: El auto envía el saldo que dispone
RECARGAR_SALDO	7	DIFIJO → DIMOVIL: Se envía el saldo que se va a recargar en el dispositivo del auto
COBRAR_PEAJE	8	DIFIJO → DIMOVIL: Se establece el cobro de peaje
NUEVAS_TARIFAS	9	PC → DIFIJO: Para establecer nuevas tarifas de cobro
SOLICITUD_ID_CASETA	10	DIFIJO → PC: Solicitando cual es el identificativo del DIFIJO
RESPUESTA_ID_CASETA	11	DIFIJO → PC: Se envía el identificativo de la caseta
ACK	12	DIMOVIL → DIFIJO: Confirmación que envía el auto a la caseta informando que el proceso fue un éxito.

Tabla 2.2 Tipo de Comando

Se tiene tres tipos de procesos en este sistema: Proceso de Cobro, Proceso de recarga y proceso de consulta de saldo.

Es necesario sólo 1 byte de comando para que los módulos puedan reconocer la función que debe realizar cuando recibe cierta trama. La tabla 2.2 se enumera los diferentes tipos de comandos que se tienen en la comunicación entre caseta y auto.

Proceso de Cobro de peaje.- En la figura 2.2 se observa el proceso en el cobro del peaje.

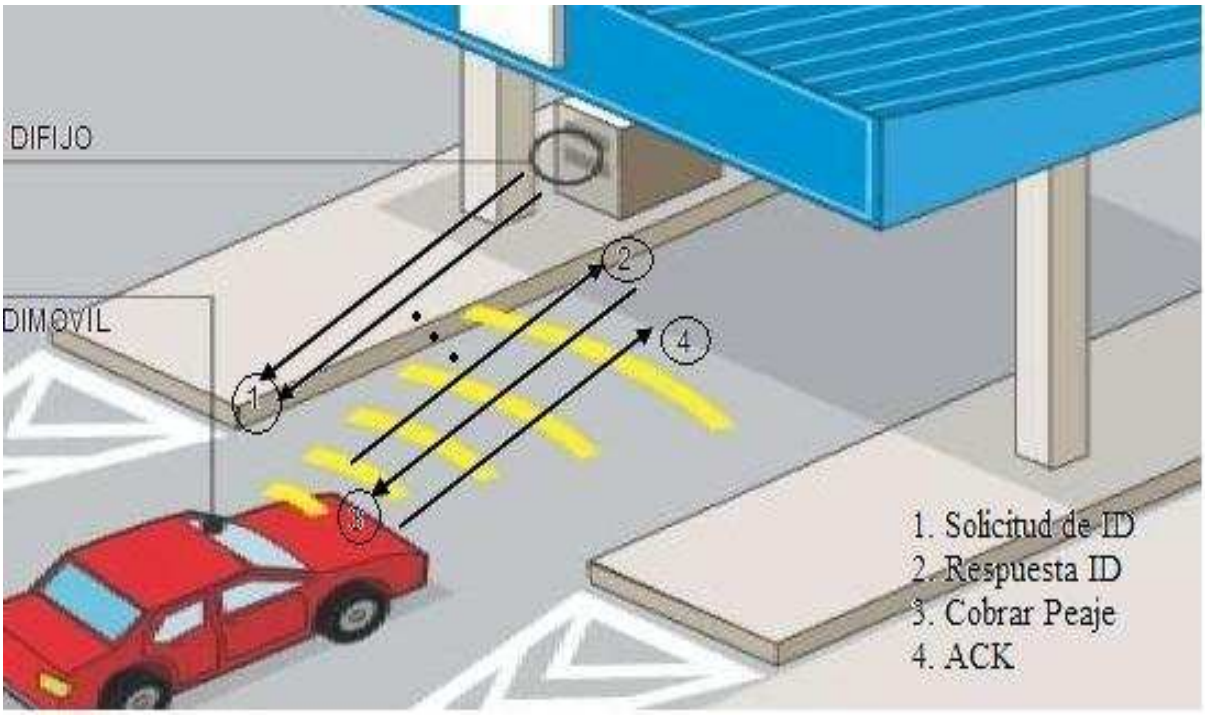


Figura 2.2 Comunicación DIFIJO y DIMOVIL

El DIFIJO está constantemente enviando una “Solicitud de ID”, cuando el auto se acerca a la estación de peaje responde con la trama “Respuesta ID”. Se procede al cobro del peaje y cuando ya se realiza el descuento, el auto envía una trama de confirmación (Trama ACK).

La trama **Solicitud_ID** en el campo ID_DESTINO envía un broadcast (11111) dirigido a todos los autos.

La trama **Respuesta_ID** es enviado por el auto, en el cual en el campo de Datos está el identificativo, en este caso la placa del auto

La trama **Cobro_Peaje** lleva en el campo de datos el valor que hay que descontar al DIMOVIL

La trama **ACK** es una confirmación de que el proceso se realizó con éxito

Proceso de Recarga

En este proceso Es similar al proceso de cobro.

- Trama 1: Solicitud de ID
- Trama 2: Respuesta de ID
- Trama 3: Recargar Saldo
- Trama4: ACK

Proceso de Consulta de Saldo

- Trama 1: Solicitud de ID
- Trama 2: Respuesta de ID
- Trama 3: Recargar Saldo
- Trama4: ACK

2.3 DISEÑO DEL HARDWARE DEL SISTEMA DE COBRO

2.3.1 DISPOSITIVO FIJO (DIFIJO)

En la figura 2.3 se observa los diferentes componentes del DIFIJO como son: La conexión USB que sirve para conectar la PC al módulo, la programación In Circuit para programar a microcontrolador si es que se necesita realizar algún cambio, El Transmisor y Receptor que transmite y recibe los datos del DIMOVIL y la señalización que nos permite visualizar los procesos en el cobro de peaje.

La parte fundamental del dispositivo es el microcontrolador, este procesa los datos que recibe tanto del DIMOVIL como de la PC.

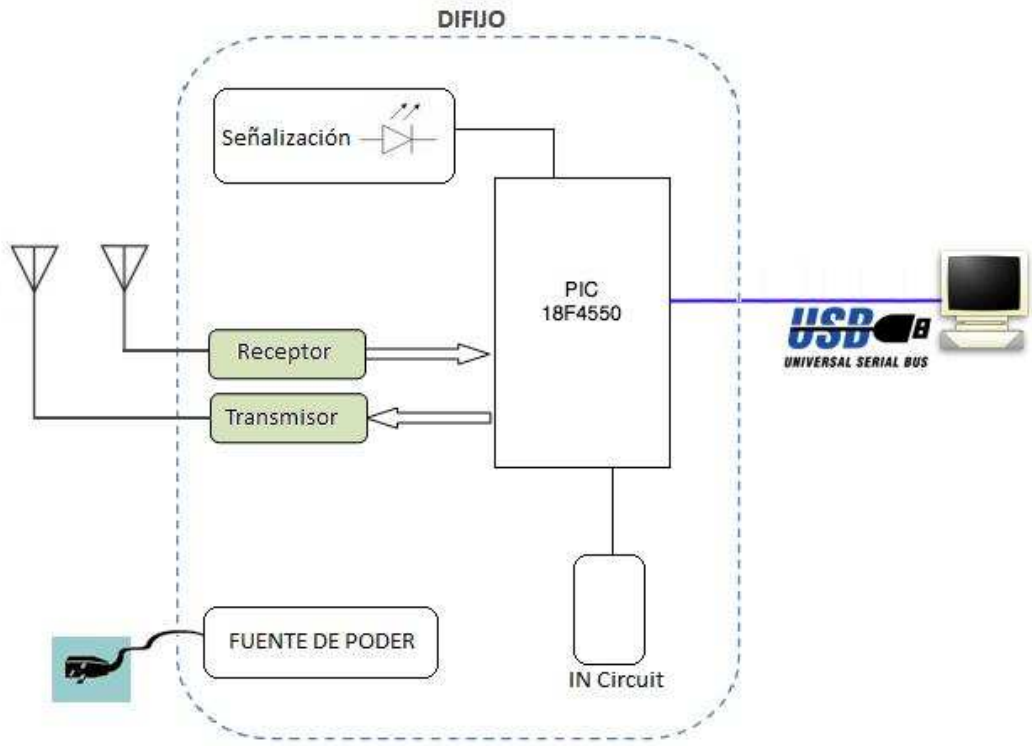


Figura 2.3 Componentes Del DIFIJO

2.3.1.1 Bloques del sistema

Fuente de poder

El diseño de la fuente de poder del DIFIJO, se observa en la figura 2.4. El voltaje de entrada a este bloque puede estar en el rango de 9 a 12 Voltios DC y tiene dos salidas: 5V y 9V. El voltaje de 5V es el que alimenta a todos los componentes del circuito, los 9V es opcional, se puede utilizar para alimentar al módulo transmisor si se quiere dar mayor potencia de salida.

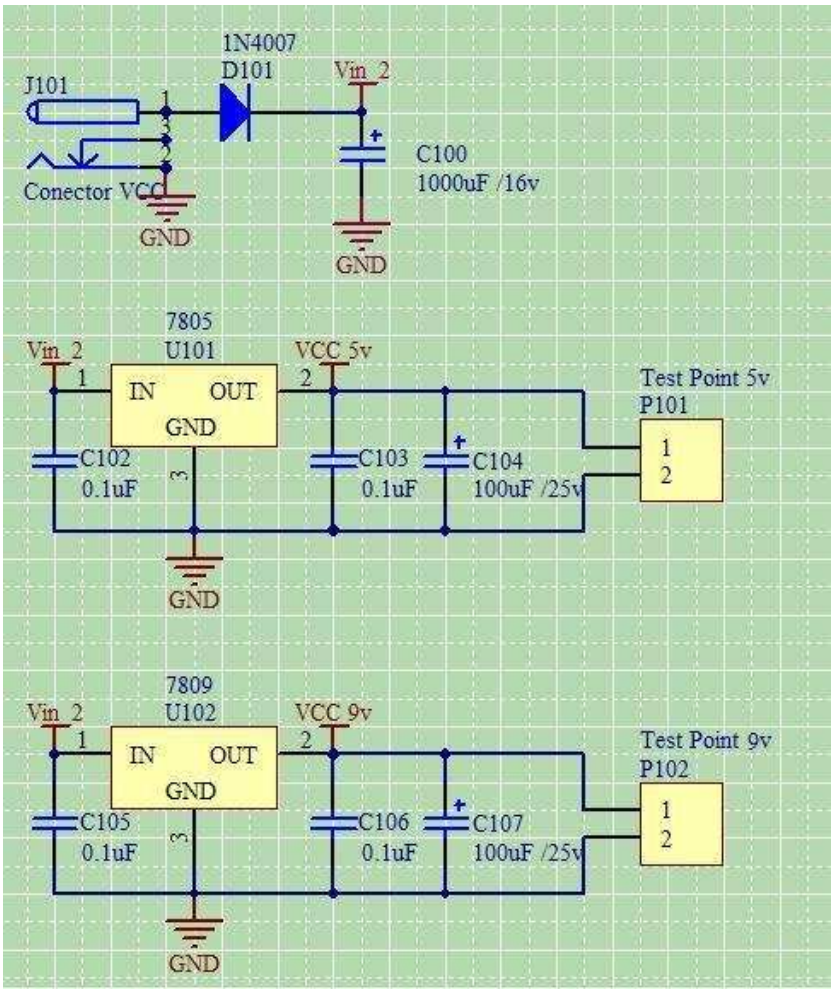


Figura 2.4 Diagrama fuente de poder

Los elementos que componen a este bloque son:

- Conector DC Jacks

Figura 2.5 Conector DC Jacks

- Diodo D101.- Protege al regulador contra alguna regresión de voltaje, algún corto circuito o por la inversión de voltaje de la fuente. Se escoge el Diodo 1N4007 para soportar una corriente de 1 A y 1000V.

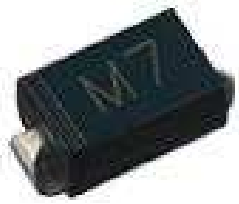


Figura 2.6 Diodo 1N4007_encapsulación SMD

- Capacitor C100.- Debe ser polarizado y de 1000 µF y al menos 16V para soportar los 12 voltios que entrega el transformador, este capacitor elimina cualquier onda para que el regulador reciba un voltaje de entrada lineal.

Figura 2.7 Capacitor electrolítico

- Regulador de voltaje 7805 y 7809.- Estos circuitos integrados tiene una salida de voltaje de 5 y 9 V_{DC} respectivamente.

Figura 2.8 CI 7805

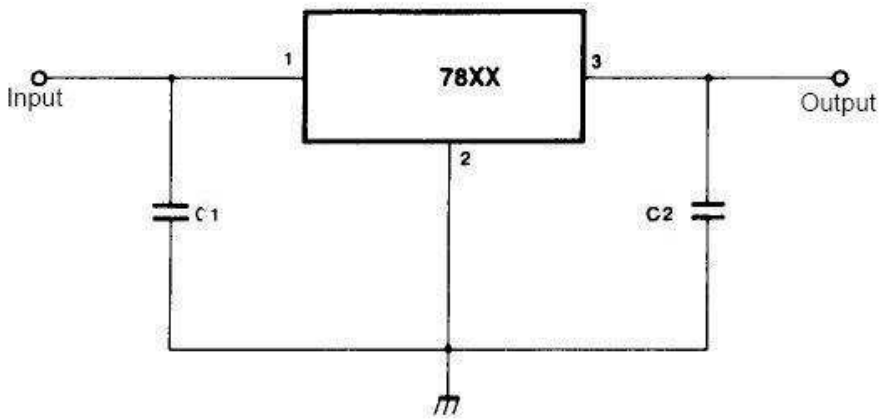


Figura 2.9 Conexión del CI 78XX

La figura 2.9 muestra la conexión que los fabricantes recomiendan para los reguladores de voltaje serie 78XX, donde XX representa el valor del voltaje de salida.

El C_1 es opcional, se suele poner cuando la distancia del alambre que conecta al circuito regulador es grande, ya que este alambre puede convertirse en

antena. El C_2 tiene como función disipar el ruido que se produce internamente en el regulador.

- Capacitores C102, C103, C105 y C106.- Elimina cualquier fluctuación de voltaje, debe ser no polarizado y de 0.1 µF (Recomendado por el fabricante).

Figura 2.10 Capacitor cerámico_SMD

- Capacitor C104 y C107.- Capacitor electrolítico de 100 µF, actúa como un balanceador de carga para asegurar una salida consistente del regulador. El valor puede estar entre 100 y 1000 µF (recomendado por los fabricantes de los 78XX)

- Jumper.- Al módulo transmisor se le alimenta con 5V o 9V y lo podemos seleccionar mediante un jumper que se encuentra en el circuito

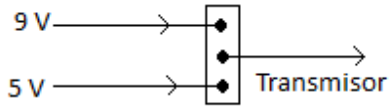


Figura 2.11 Jumper para seleccionar el voltaje

Módulo Transmisor/Receptor

La figura 2.12 muestra el diagrama de conexión del transmisor. El microcontrolador envía los datos hacia este módulo y este se encarga de enviar por el aire mediante radiofrecuencia.

Los datos que llegan hacia el Receptor son transformados en bits para enviar al microcontrolador. El diagrama de conexión de este módulo se observa en la figura 2.13.

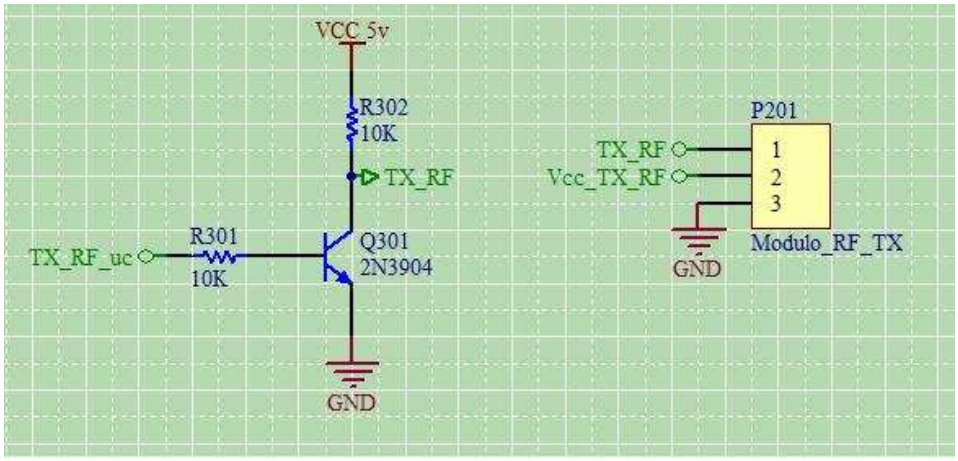


Figura 2.12 Diagrama del Transmisor

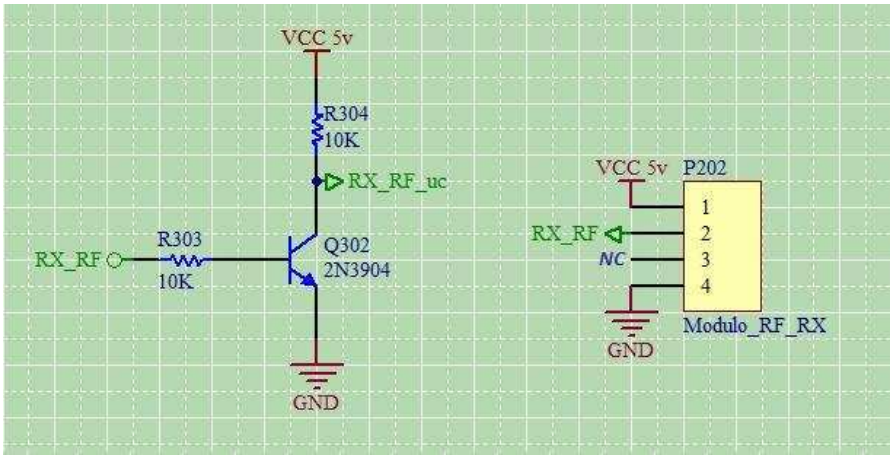


Figura 2.13 Diagrama del Receptor

Para evitar que el pórtico de recepción y transmisión del microcontrolador manejen directamente a los módulos y proteger el mismo de una sobre corriente, es conveniente usar un transistor, como se muestra en la figura 2.12 y 2.13.

En los pins del PIC se manejan niveles de voltaje TTL, para establecer la acción que realiza el programa almacenado en el mismo. Si a esta tensión se le hace caer una resistencia, se genera una corriente de acuerdo a la ley de ohm y considerando que, la corriente máxima que soporta cualquier pin, es de 25 mA[61]; se creyó conveniente establecer una corriente en de 0.5 mA y bajo esta premisa, se efectúa el diseño como sigue:

$$Rpin = \frac{V_{PIN}}{I_{PIN\,\min}}$$
$$R = \frac{5V}{0.5mA}$$
$$R = 10K\Omega$$

Posteriormente esta corriente circula por la base de un transistor NPN 2N3904, el cual, trabajando en corte y saturación, actúa a modo de switch haciendo circular corriente o no, de acuerdo a lo que entregue el pin del microcontrolador.

En el colector, una corriente de 0.5 mA, es suficiente para que el transistor se sature. Ya que el circuito se polariza con 5V, esto da como resultado que las resistencias de colector tengan un valor de 10KΩ.

$$Rc = \frac{V}{I_{Csaturación}}$$
$$R = \frac{5V}{0.5mA}$$
$$R = 10K\Omega$$

[61] Data Sheet PIC18F2455/2550/4455/4550

Programación In Circuit

Para programar el microcontrolador se utiliza el PICkit 2: Es una herramienta fabricado por Microchip© para programar toda su línea de microcontroladores PIC's® desde los PIC10, PIC12, PIC14, PIC16, PIC18, PIC24, dsPIC30 y dsPIC33. El programador fue diseñado para programar los microcontroladores en circuito (ICSP) lo que significa que se puede programar los microcontroladores montados directamente en la aplicación y/o protoboard sin necesidad de tener que sacarlo y meterlo cada vez que se modifica el programa.

En la figura 2.14 se muestra al programador utilizado y en la figura 2.15 la descripción de los pins.

Legend:

1 – Status LEDs	3 – Lanyard Connection	5 – Pin 1 Marker
2 – Push Button	4 – USB Port Connection	6 – Programming Connector

Figura 2.14 Programador PICkit2[62]

[62] PICkit2 User's Guide

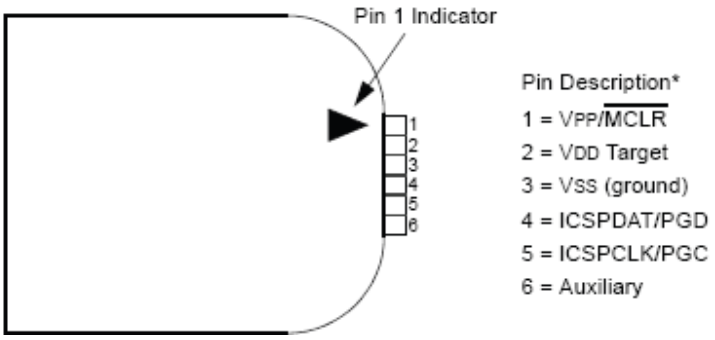


Figura 2.15 Pins del programador PICkit2[63]

Figura 2.16 PICkit2 Completo

El cable USB que se observa en la figura 2.16 sirve para conectar a la PC y cargar en el programador el código que se va a grabar en el microcontrolador.

La figura 2.17 muestra la conexión In Circuit que se implementa en la placa del DIFIJO.

[63] PICkit2 User's Guide

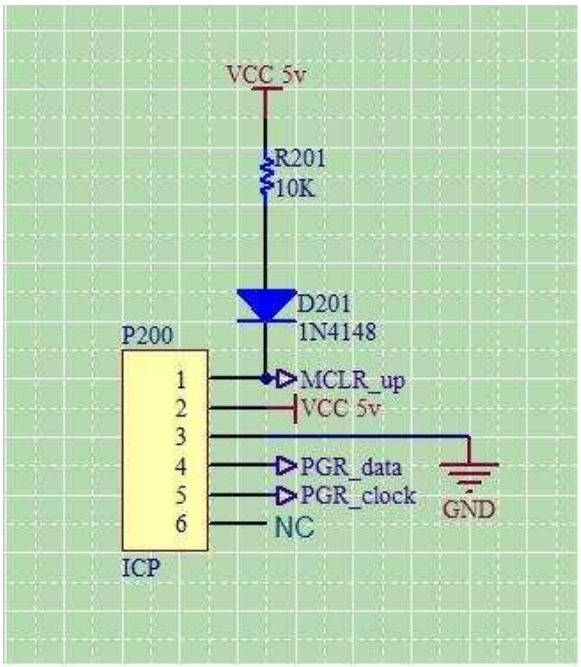


Figura 2.17 Diagrama In Circuit

El pin 18 ($\overline{\text{MCLR}}$/V_{PP}/RE3) del microcontrolador puede ser utilizado para reset ($\overline{\text{MCLR}}$) o como un puerto de entrada y salida, por tal motivo se coloca el diodo 1N4148 y la resistencia de 10KΩ, que sirven como un circuito de aislamiento cuando un voltaje de programación (12V) es aplicado.

Conexión USB

La figura 2.18 muestra la conectividad del USB al microcontrolador. Se observa que los dos pins de afuera son los encargados de proveer alimentación al dispositivo conectado al puerto. Hay que tener en cuenta que la tensión de salida de los USB es de 5V, y la máxima intensidad de corriente que puede entregar es 500 mA.

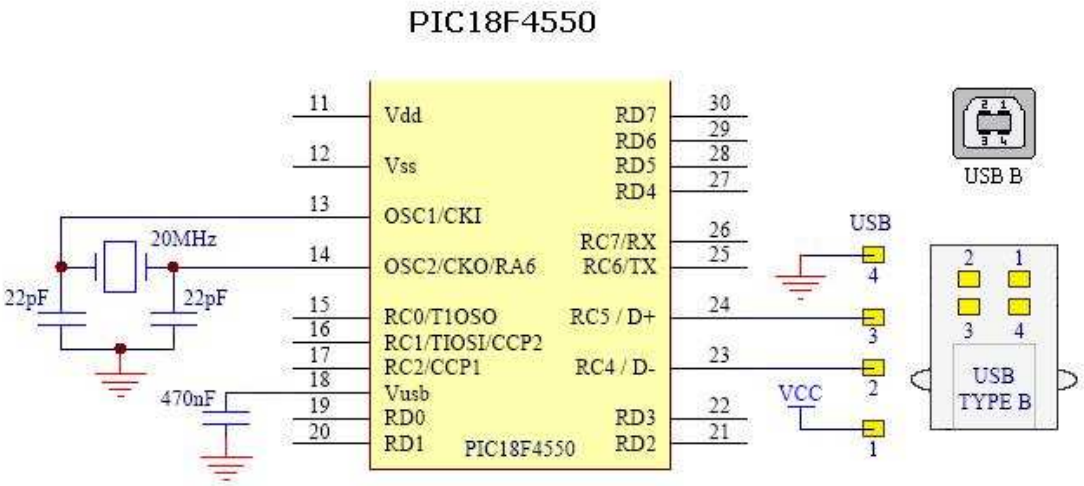


Figura 2.18 Conexión USB

El DIFIJO cuenta con su propia alimentación, por lo tanto, el voltaje del USB se envía a tierra mediante una resistencia.

V=5V (Tensión de salida de los USB)

I=500mA (Máxima corriente que puede entregar los USB)

$$R \geq \frac{V}{I}$$

$$R \geq \frac{5V}{500mA}$$

$$R \geq 10\Omega$$

El microcontrolador detecta la conexión USB mediante el Pin 10 (VMO).

$$R \geq \frac{V}{I}$$

$$R \geq \frac{5V}{500mA}$$

$$R \geq 10\Omega$$

Se escoge una resistencia de 100 KΩ, para ambos casos para que la corriente del USB sea mínima. Con ese valor de Resistencia la corriente es 100µA.

El condensador que va en el pin V_{USB} del micro sirve para regular la tensión USB que recibe de los puertos de la PC, para filtrado y estabilización del voltaje del regulador interno de 3.3V que trae el PIC18F4550[64]. La figura 2.19 muestra el diagrama que se implementa en el circuito.

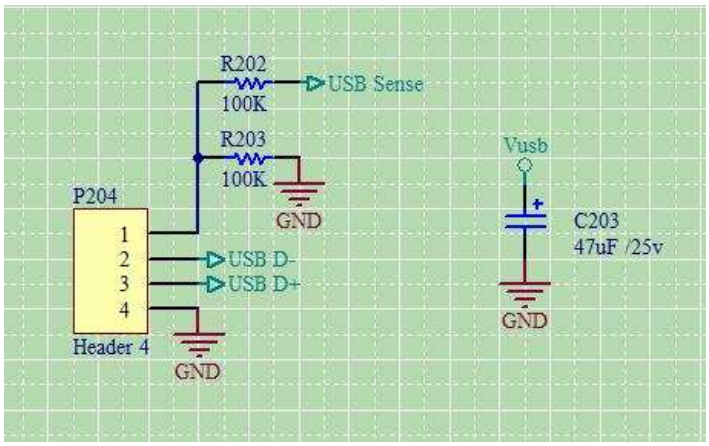


Figura 2.19 Diagrama de Conexión

Señalización

Para la señalización del DIFIJO se dispone de 3 LEDs: Rojo, Naranja y Verde, la función de cada uno se detalla en la tabla 2.3 y en la tabla 2.4 se observa las características de cada uno.

LED	FUNCIÓN
ROJO	Permanece siempre prendido. Se apaga cuando se procesa el cobro de peaje
NARANJA	Titila cuando está enviando tramas
VERDE	Se enciende si se produjo el cobro de peaje

Tabla 2.3 Señalización DIFIJO

[64] Data Sheet PIC18F2455/2550/4455/4550

Color	Intensidad Máxima (mA)	Intensidad media (mA)	Caída de Tensión (V)	Diámetro (mm)
Rojo	20	5 - 10	1.6	3
Naranja	20	5 – 10	1.7	3
Verde	20	5 - 10	2.4	3

Tabla 2.4 Características de los LEDs[65]

En la figura 2.20 se encuentra la implementación en la placa.

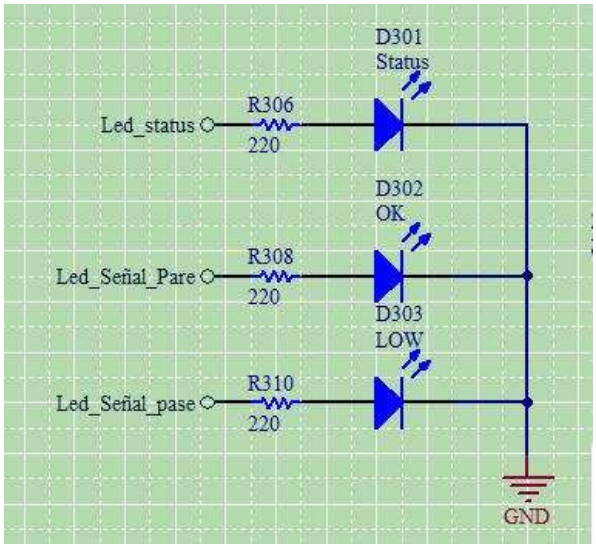


Figura 2.20 Diagrama de Señalización del DIFIJO

Cálculo de la resistencias colocada en serie con el LED Rojo.

V = 5V, voltaje suministrado por el micro (1_L)

I= 20 mA, Intensidad de corriente máxima que soporta el LED

[65] http://www.superpbenavides.com/catalogo/componentes%20pasivos/optoelectronicos/Como-disenar-LED.pdf

$$R = \frac{V - V_{LED}}{I}$$

$$R = \frac{5 - 1.6}{20} K\Omega$$

$$R \geq 0.17K\Omega \rightarrow R = 220\Omega$$

El cálculo anterior se realiza tomando como referencia la corriente máxima que puede soportar el LED. Se escoge el mismo valor de resistencia para los 3 LEDs porque la caída de tensión de cada uno no varía significativamente.

Microcontrolador PIC18F4550-I/PT

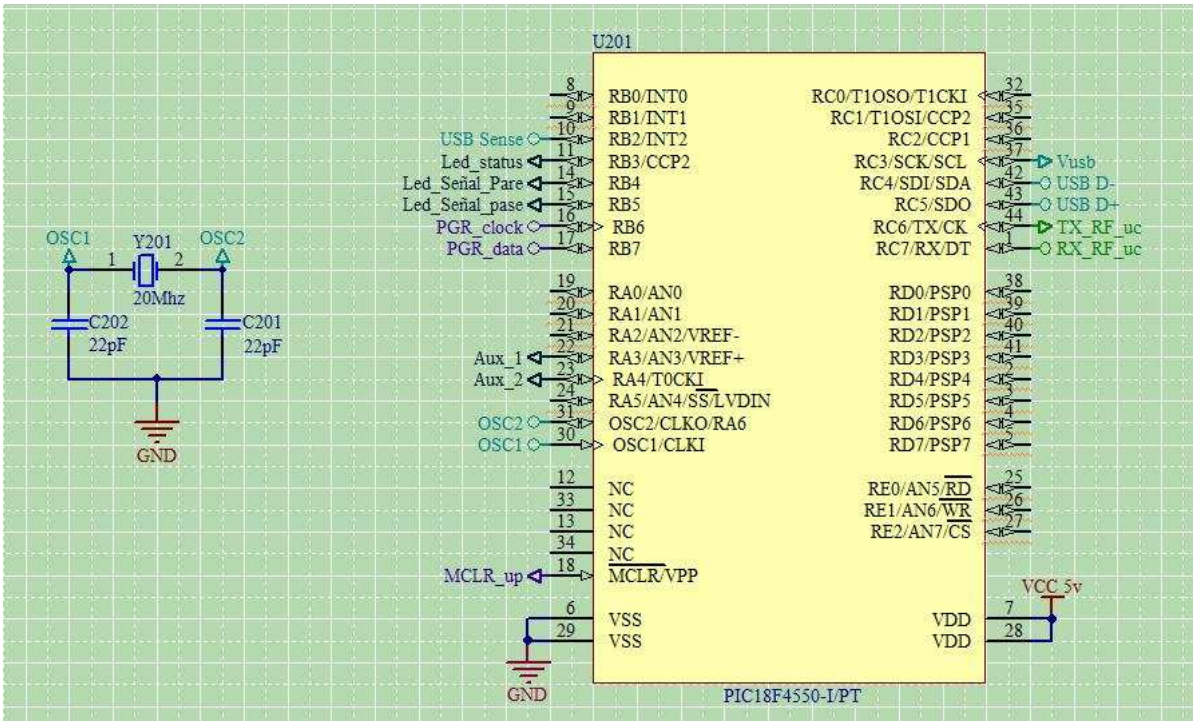


Figura 2.21 Diagrama del Microcontrolador[66]

[66] Data Sheet PIC18F2455/2550/4455/4550

Esta es la parte principal del dispositivo, el microcontrolador procesa los datos que llegan del DIMOVIL. Es el que realiza el descuento del valor del peaje al auto y envía los datos que se visualizan en la PC.

La figura 2.21 muestra el diagrama de la conexión del microprocesador, funciona con un oscilador de 20 MHz.

Módulo completo

En las figuras 2.22 a 2.26 se observa la placa (PCB) y la circuitería final para el DIFJO.

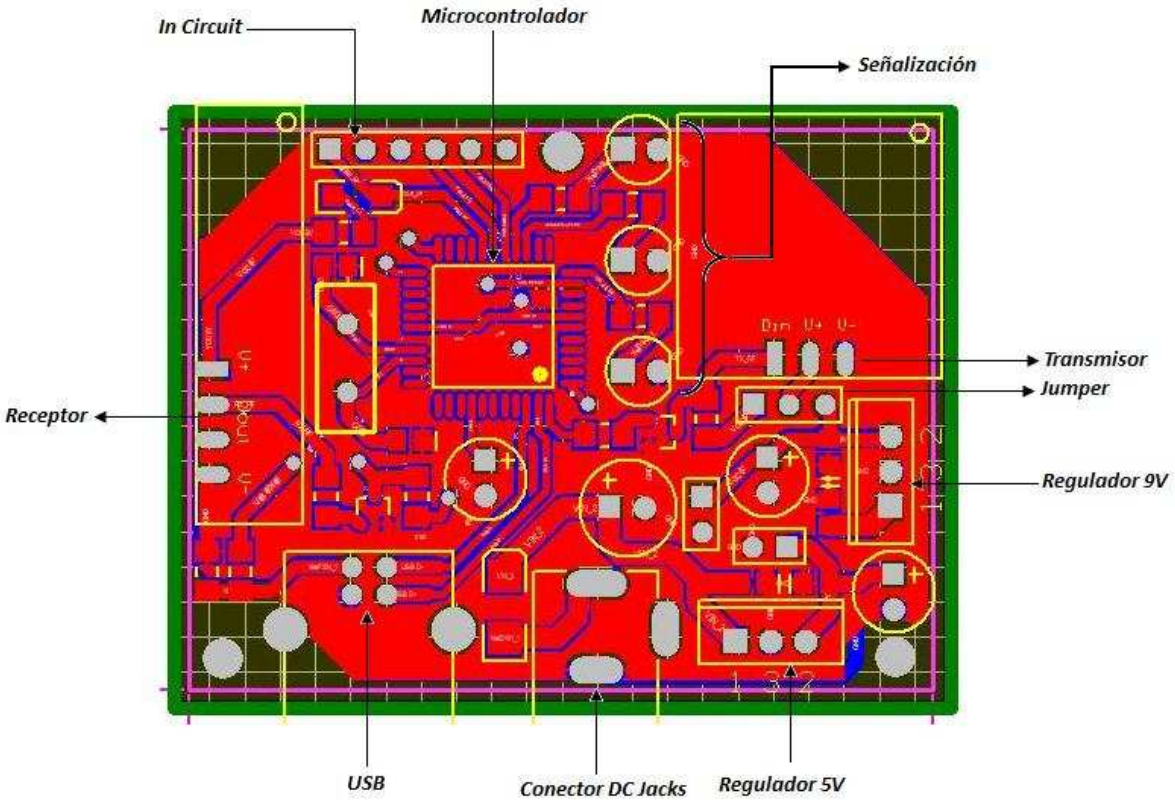


Figura 2.22 PCB DIFIJO

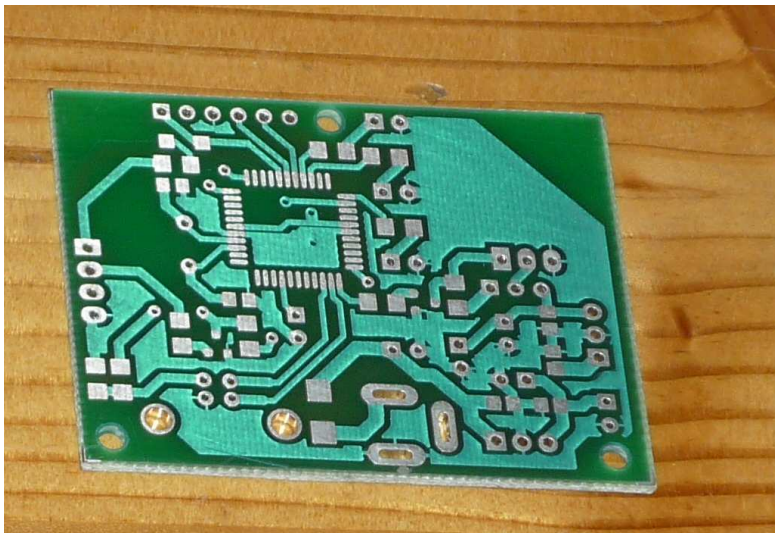


Figura 2.23 PLACA DIFIJO

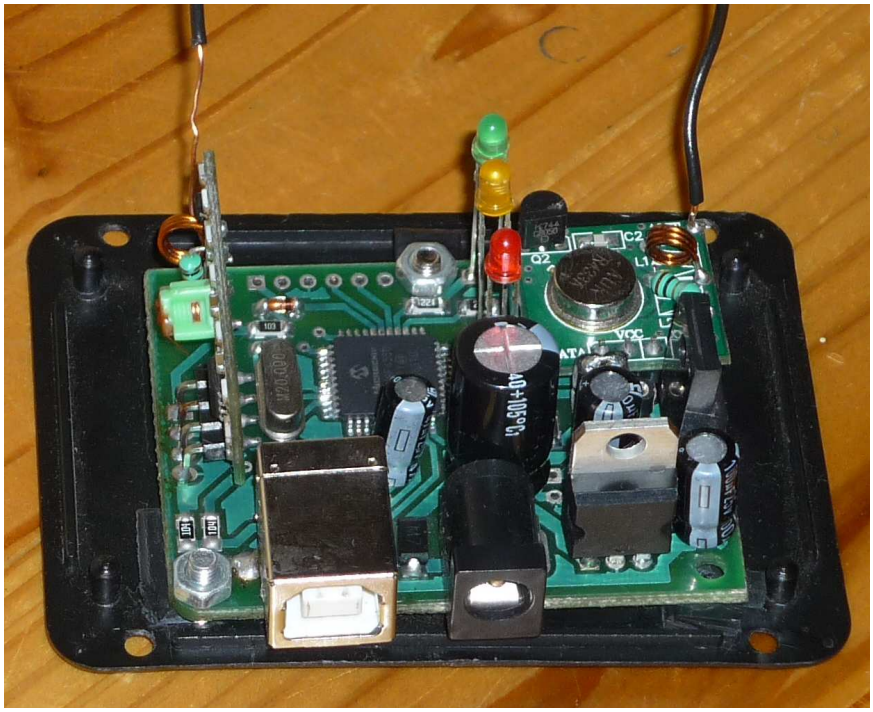


Figura 2.24 Circuito DIFIJO

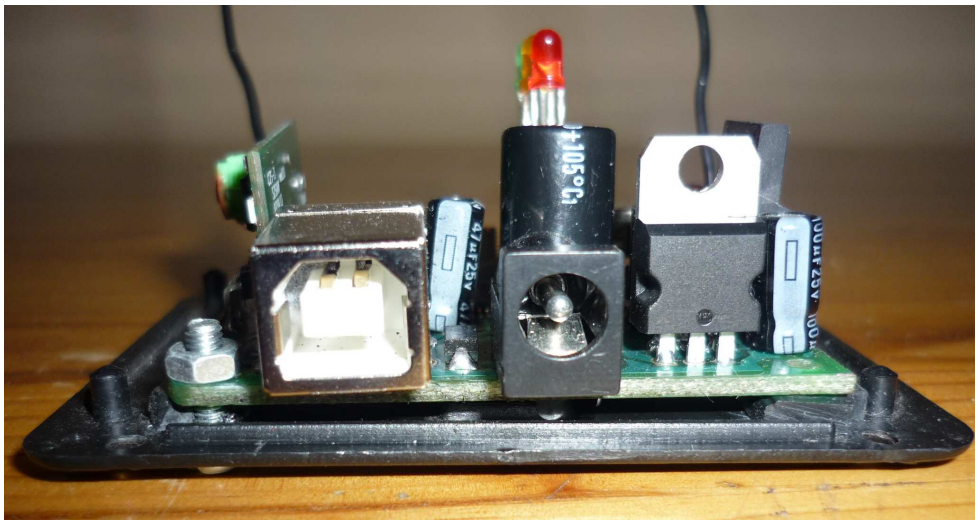


Figura 2.25 Circuito DIFIJO (Vista frontal)

Figura 2.26 Circuito DIFIJO (Con la caja)

El diagrama completo del circuito se encuentra en el ANEXO C.

2.3.2 DISPOSITIVO MÓVIL (DIMOVIL)

En la figura 2.27 se muestra los componentes del DIMOVIL. Consta de un microcontrolador que se encarga del funcionamiento del dispositivo. El transmisor y receptor para establecer la comunicación con el DIFIJO, el bloque In Circuit sirve para programar al microcontrolador si se quiere realizar algún cambio, una memoria EEPRON externa que contiene el saldo del auto y tres LEDs (Verde, Rojo y Naranja) para señalización.

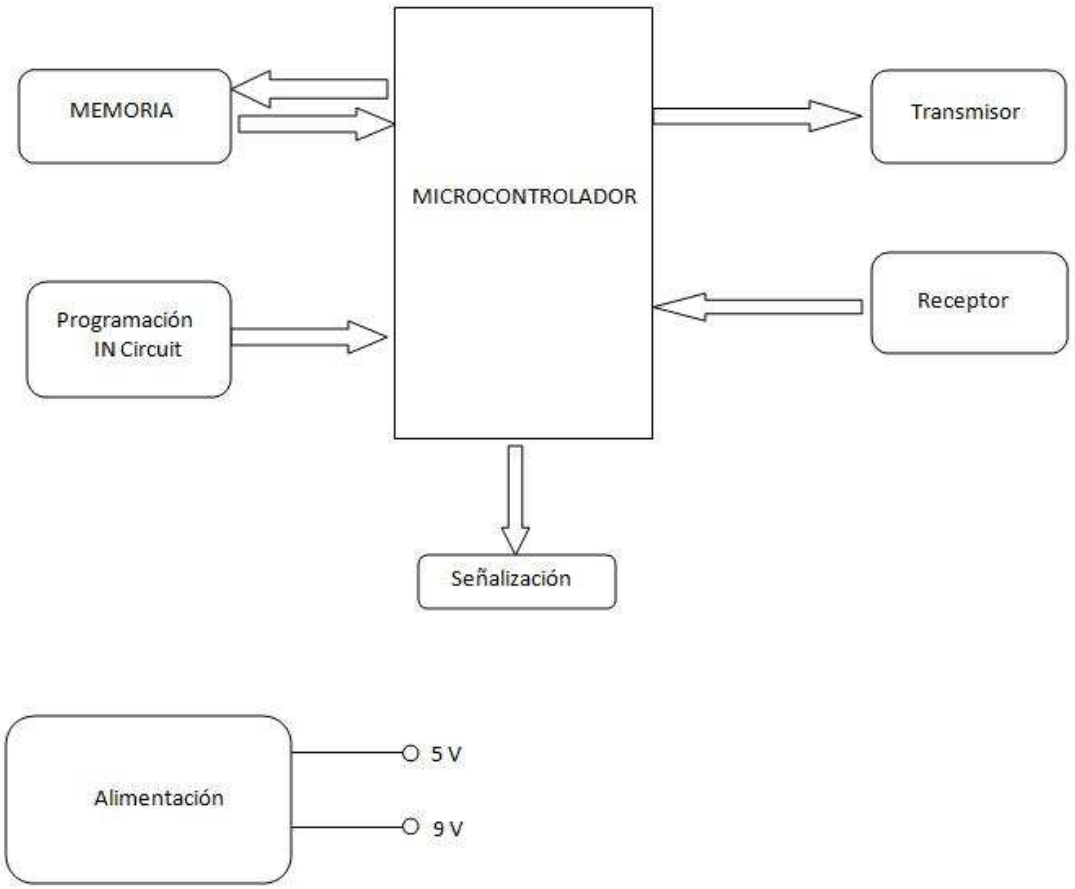


Figura 2.27 Componentes del DIMOVIL

2.3.2.1 Bloques del sistema

El diseño de la fuente de poder, LEDs de señalización, Programación In Circuit y el bloque para recepción y transmisión de datos es similar al DIFIJO.

Fuente de poder

En la figura 2.28 se observa el diagrama de la fuente de poder del DIMOVIL.

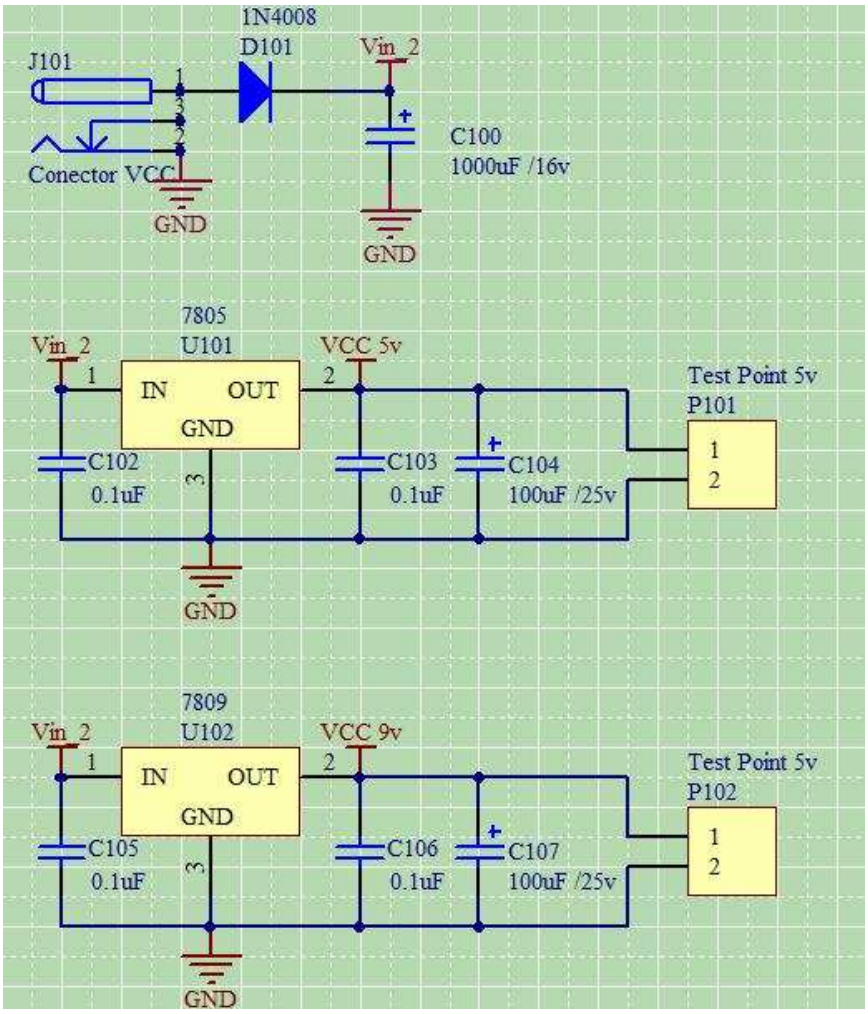


Figura 2.28 Fuente de energía DIMOVIL

Transmisor/Receptor

Estos módulos sirven para intercambiar datos con el DIFIJO. La figura 2.29 y 2.30 muestran los diagramas de conexión de estos módulos al microcontrolador.

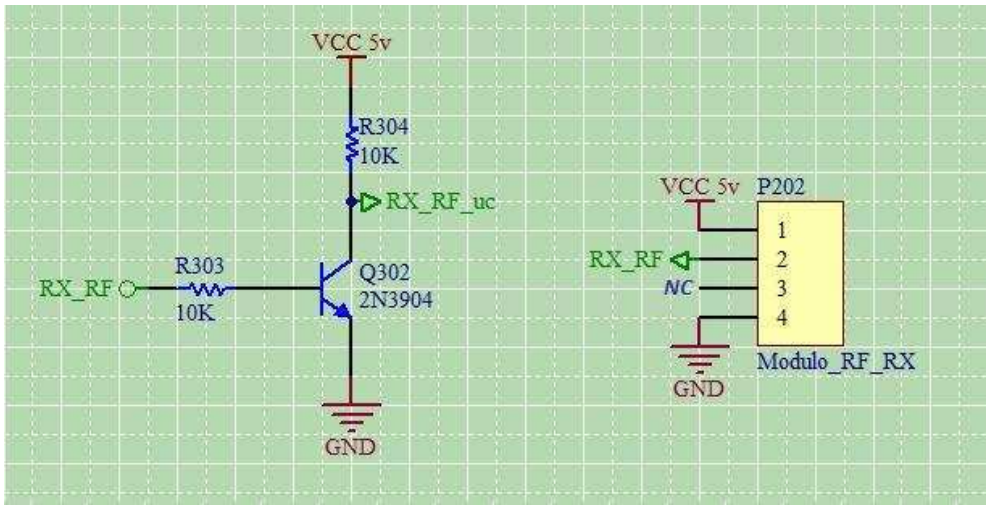


Figura 2.29 Diagrama Receptor

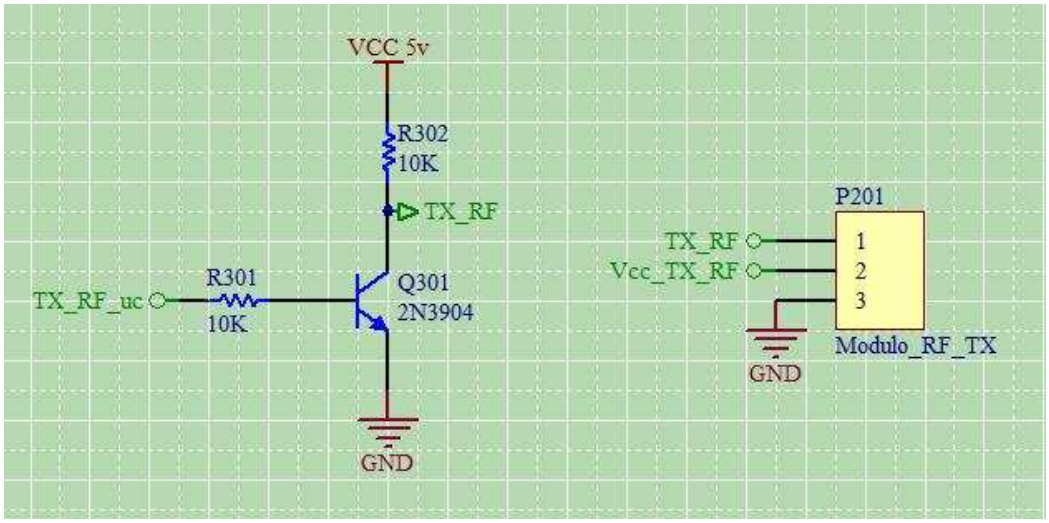


Figura 2.30 Diagrama Transmisor

Programación In Circuit

El bloque de programación In Circuit se utiliza para programar el microcontrolador. En la figura 2.31 se observa el diagrama de conexión al microcontrolador.

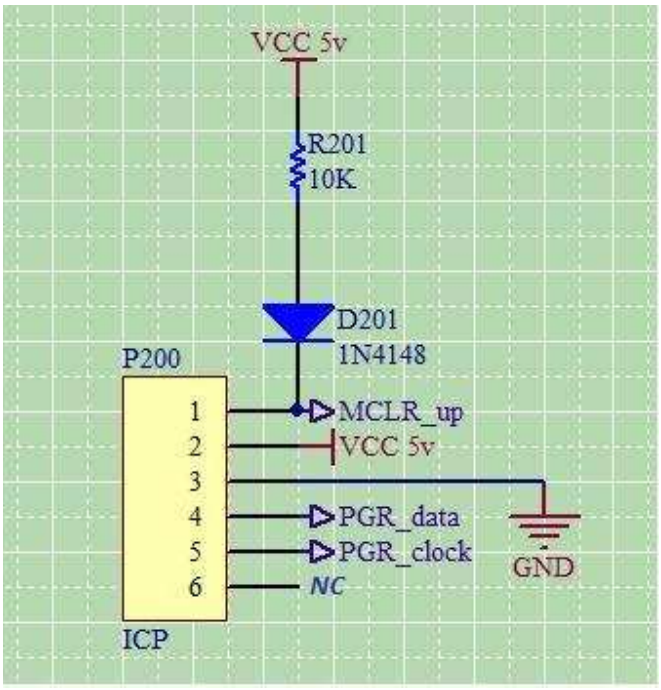


Figura 2.31 Programación In Circuit

Señalización

Para la señalización del DIMOVIL se dispone de 3 LEDs: Rojo, Naranja y Verde, la función de cada uno se detalla en la tabla 2.5 y la figura 2.32 muestra el diagrama de conexión.

LED	FUNCIÓN
ROJO	Se enciende cuando tiene poco saldo, para nuestro caso un saldo menor a 1 $
NARANJA	Titila cuando recibe o envía tramas
VERDE	Está prendido si su saldo es mayor a 1 Dólar

Tabla 2.5 Señalización DIMOVIL

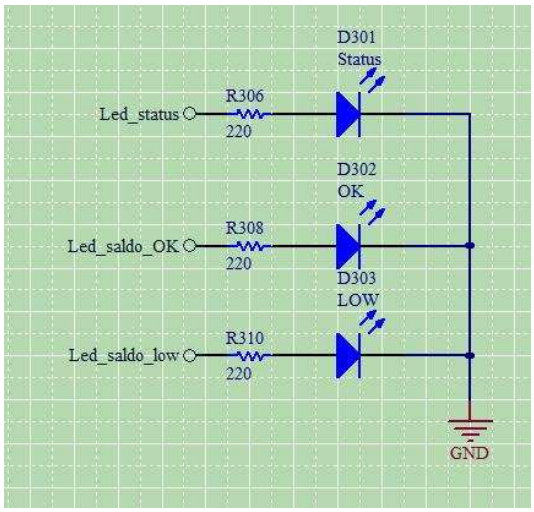


Figura 2.32 Señalización del DIMOVIL

Memoria

En la figura 2.33 se observa el diagrama de conexión de la memoria EEPRON. Los terminales SDA y SCL son de colector-abierto por tal motivo requieren resistencias de pull-up hacia Vcc, típicamente de 10 KΩ para 100 KHz.

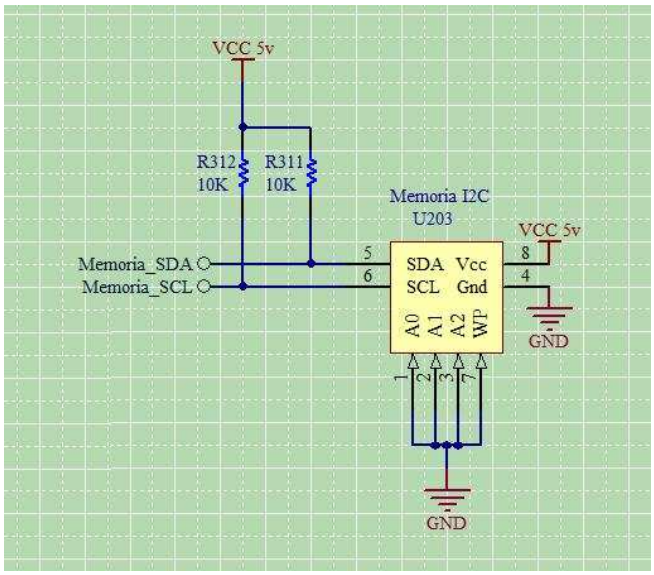


Figura 2.33 Diagrama de conexión de la memoria

Las resistencias de pull up mantienen las líneas en nivel alto cuando el bus está libre y asegura que la señal sea levantada del nivel bajo al alto.

Microcontrolador

En el DIFIJO no se requiere de una conexión USB, por tal motivo se va a utilizar un microcontrolador de menores características, en el presente proyecto se utiliza el modelo PIC16F876A-I/SO. La figura 2.34 muestra el diagrama de conexión del microcontrolador en la placa.

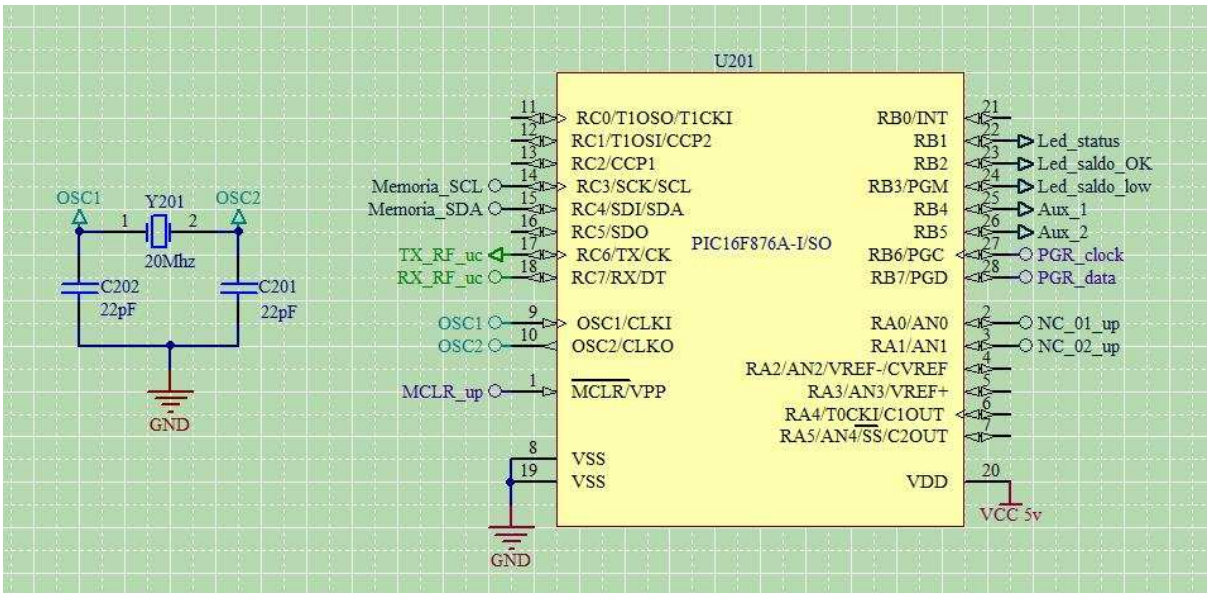


Figura 2.34 Diagrama de conexión del Microcontrolador

Módulo completo

En la figura 2.35 y 2.39 se observa el PCB del DIMOVIL y el circuito resultante para el DIMOVIL.

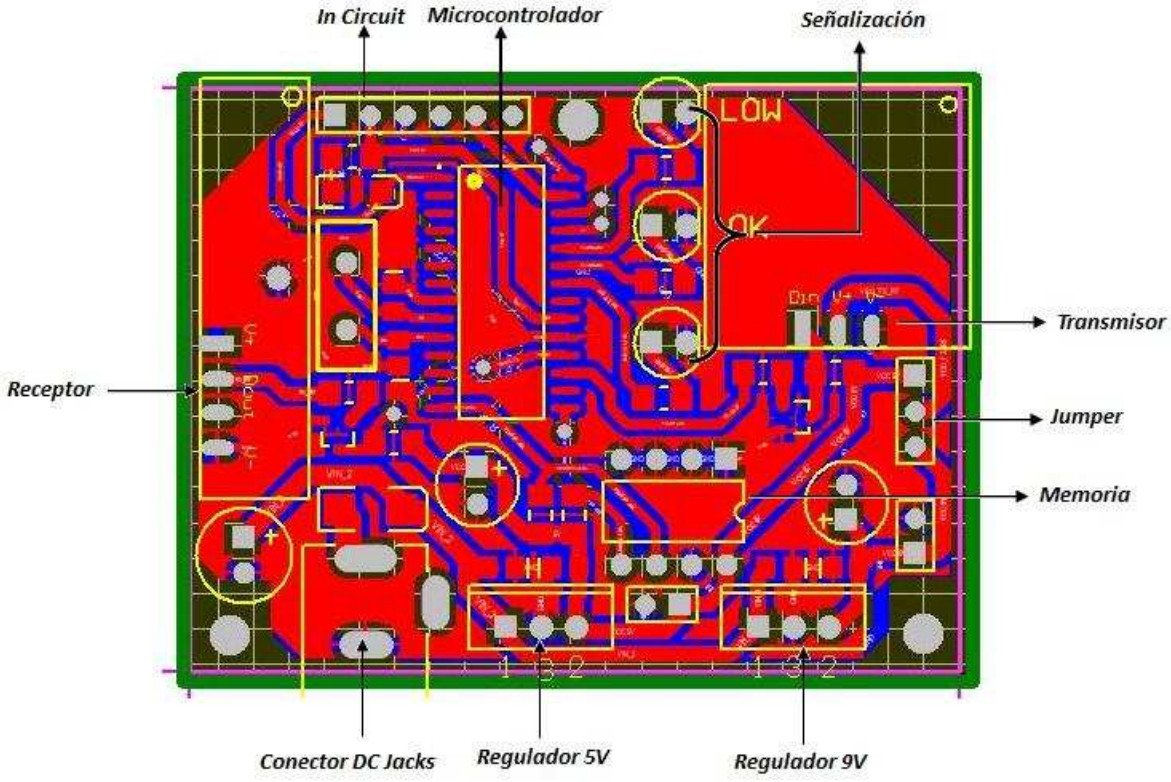


Figura 2.35 PCB DIMOVIL

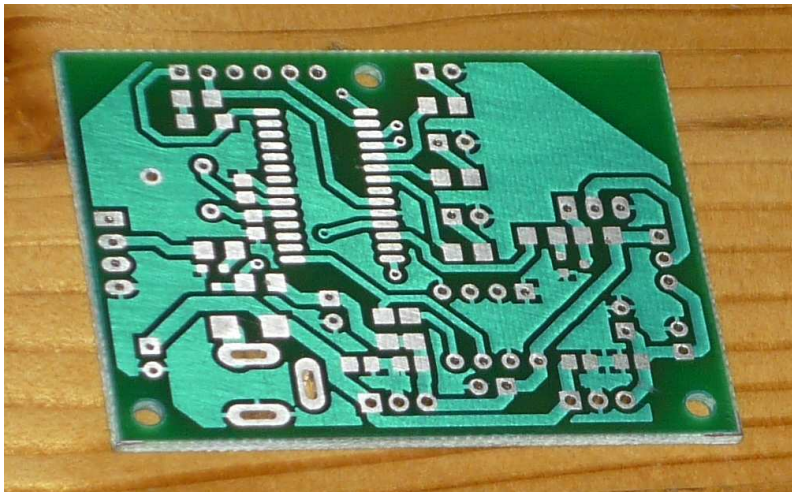


Figura 2.36 Placa del DIMOVIL

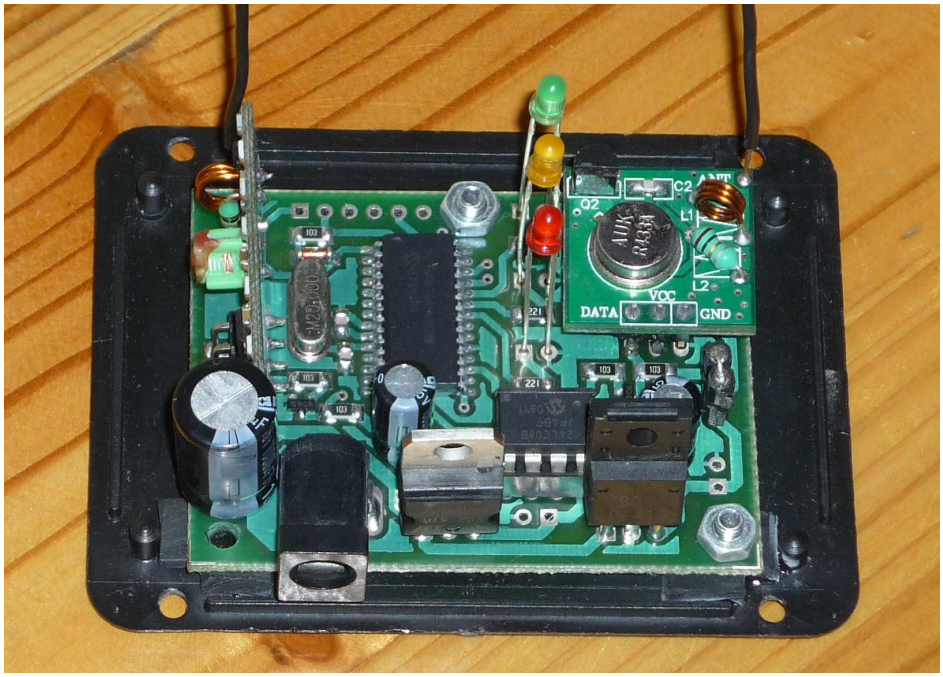


Figura 2.37 Circuito DIMOVIL

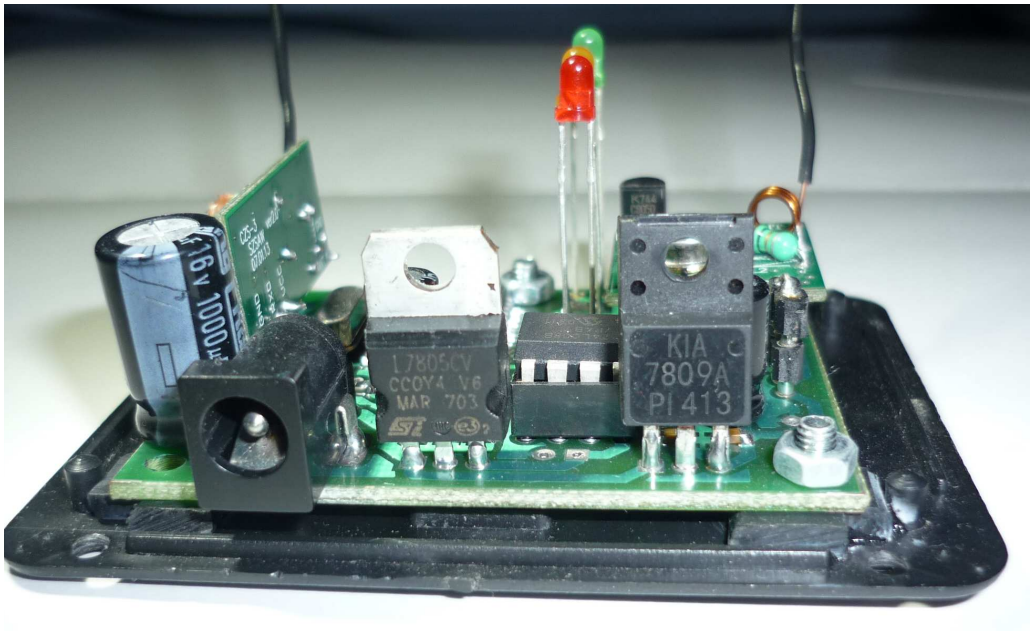


Figura 2.38 Circuito DIMOVIL (Vista Frontal)

Figura 2.39 Circuito DIMOVIL (En la Caja)

El diagrama completo se encuentra en el anexo D.

2.4 DISEÑO DEL SOFTWARE DEL SISTEMA DE COBRO

2.4.1 COMPILADOR CCS

En esta sección se describen los elementos más básicos y esenciales para comenzar a programar. El programa que controla al microcontrolador se desarrolla en lenguaje C, por lo tanto es preciso utilizar también un Compilador C. El que se va a utilizar es el PCW de la casa CCS Inc.

2.4.1.1 Entorno de trabajo

Este entorno de trabajo permite compilar y suministra una gran variedad de herramientas auxiliares. La figura 2.40 muestra la interfaz gráfica de este entorno de trabajo. Existen dos formas de iniciar una sesión: abriendo un fichero de código fuente o creando un proyecto.

Este compilador traduce el código C del archivo fuente (.C) a lenguaje de máquina para los microcontroladores PIC, generando así un archivo en formato hexadecimal (.HEX).

Para abrir un fichero fuente directamente se realiza una pulsación sobre el ícono para el manejo de ficheros y aparece un menú donde se puede crear, abrir, guardar o cerrar ficheros. Con el comando NEW se puede crear un fichero fuente, un proyecto, un fichero RTF o un fichero de diagrama de flujo (figura 2.41).

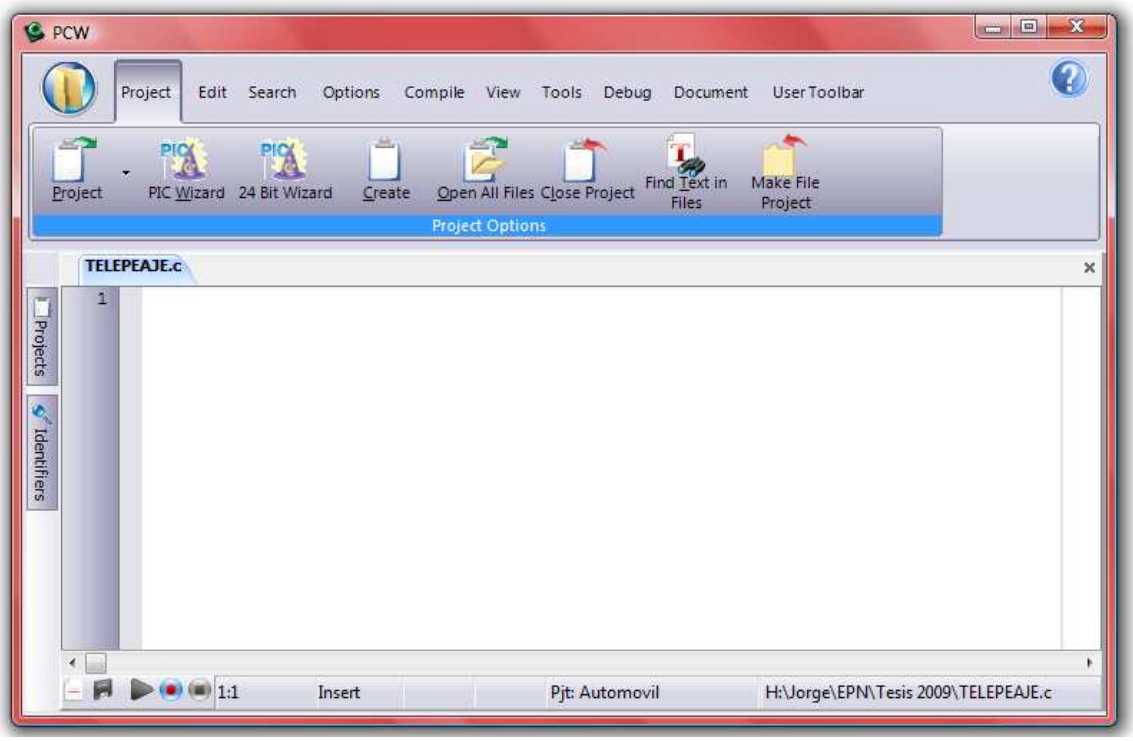


Figura 2.40 Entorno de trabajo PCW

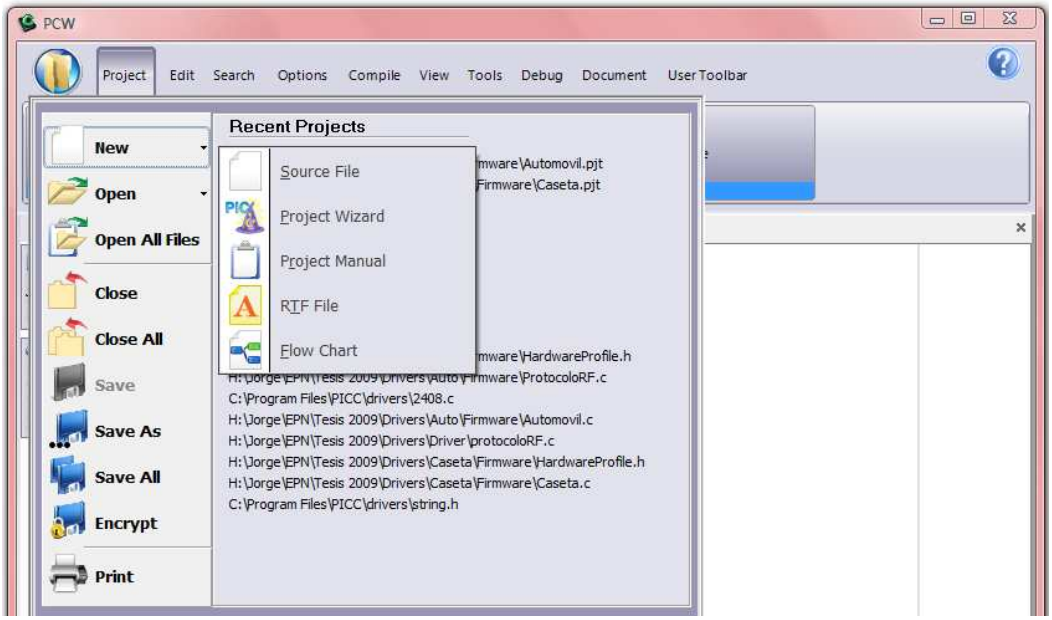


Figura 2.41 Los Menús Para el Manejo de los Ficheros

Opción NEW → Source File, el programa pide el nombre del nuevo fichero y crea una nueva ventana en blanco donde se puede empezar a escribir el código.

Opción NEW → Project Wizard, tras pedir el nombre del nuevo proyecto aparece la ventana de configuración con dos pestañas, una para configurar las distintas opciones que se muestran en la barra izquierda (Figura 2.42) y otra donde se muestra el código resultante de la configuración. Recorriendo las distintas opciones (general, communications, etc) se llega a obtener el código de la configuración deseada, tras lo cual ya se puede empezar a escribir el resto del código del programa. Se debe observar cómo se incluye un fichero de cabecera *.h (figura 2.43) donde se encuentra la configuración del dispositivo.

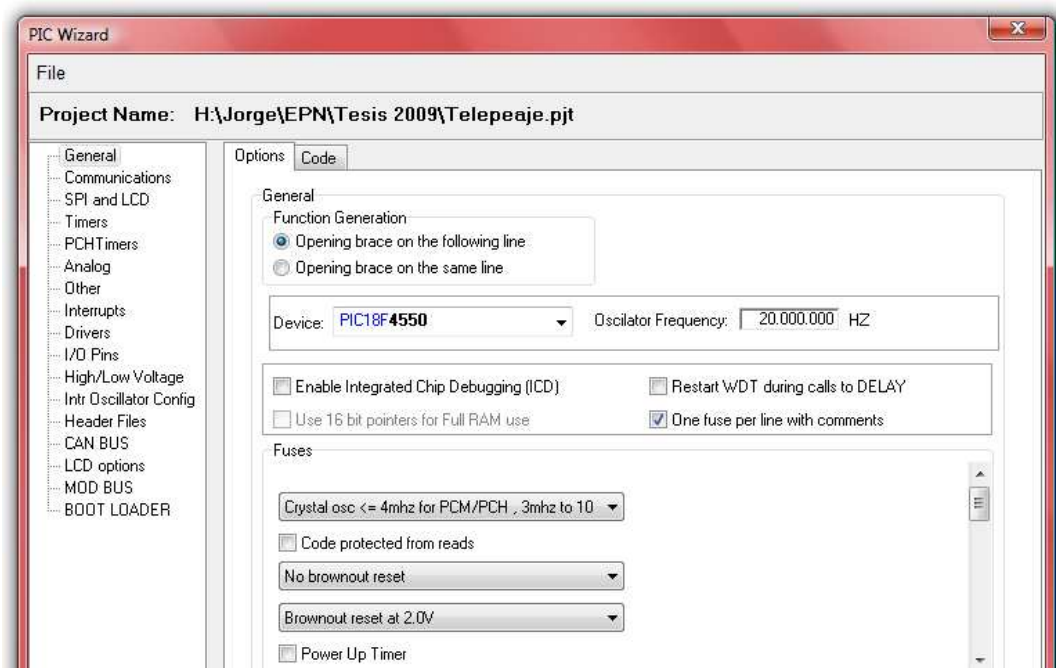


Figura 2.42 Ventana de Configuraciones de las Opciones

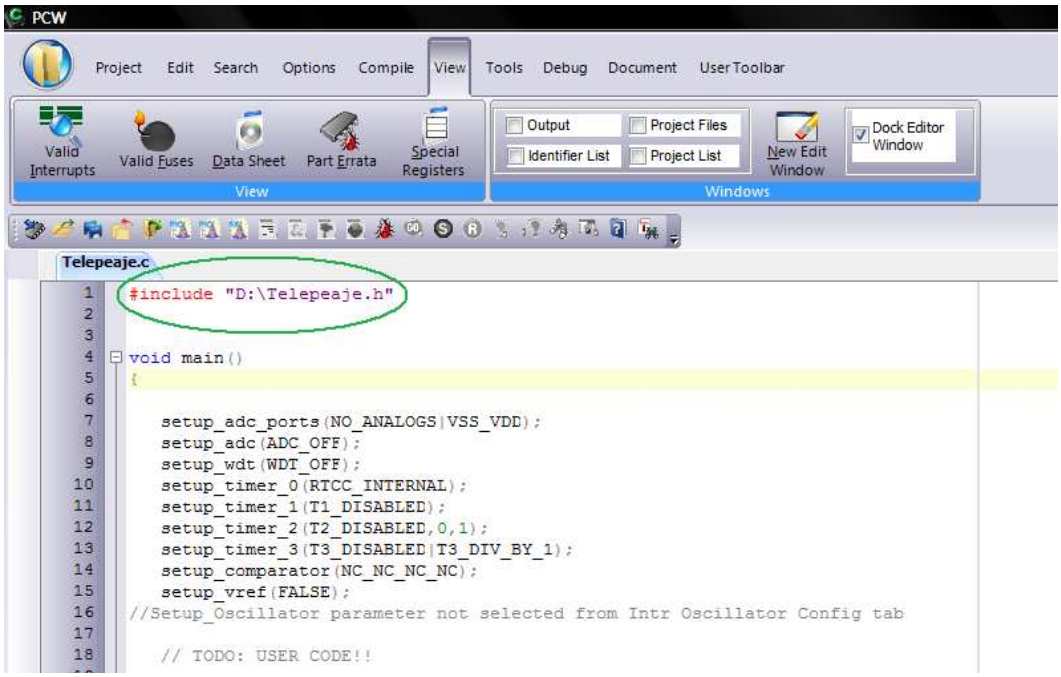


Figura 2.43 Código resultante

2.4.1.2 Funciones

Se define una función como un conjunto de sentencias **C** que, a partir de unos argumentos de entrada, realizan una determinada tarea, y devuelven un valor de salida.

La función está caracterizada por su tipo (puede ser cualquiera, incluso un puntero), su nombre y la lista de parámetros que recibe. Como todo bloque de sentencias, las correspondientes a una función van encerradas entre llaves. Si no se indica el tipo de la función, se toma por defecto el tipo `int`, aunque se recomienda indicar el tipo siempre por prudencia y portabilidad.

El nombre puede ser cualquiera excepto el de "`main`", que se reserva para la función principal, la que inicia la ejecución del programa. La función debe definirse del tipo de dato que se quiere que devuelva como resultado. Si la función no devuelve ningún valor debería usarse el tipo `void` (ausencia de tipo). Aunque no es estrictamente necesario usarlo, se recomienda hacerlo, de la misma forma que se recomienda definir una función `int` como tal aunque sea el tipo por defecto.

Los parámetros irán precedidos por su tipo, y separados unos de otros por comas. Por ejemplo, una función que devuelve un puntero a un carácter (o una cadena) sería:

```
char *ordena(int x, char *s, float a, float b)
{
if (s[x] == 0) s++;
....
return(s);
}
```

Se permite también que una función no reciba ningún argumento (a veces, en este caso se usa también el tipo `void`, aunque es poco habitual dada la imposibilidad de ambigüedad). Ejemplo:

```
void lee_datos() { ... }

main()

{

...

lee_datos();

...

}
```

Se puede utilizar la palabra clave `return` para que una función devuelva un valor y el programa vuelva inmediatamente a la función llamante. En este caso, debe asegurarse que el tipo de la expresión devuelta coincide con el de la función.

El código de un programa C suele distribuirse entre varios ficheros fuente (con extensión .c), denominados **módulos**. Así, habitualmente un programa de cierto tamaño constará de un módulo principal que contiene, al menos, la función `main()`, otros módulos con el código de diversas funciones agrupadas según algún criterio, o con definiciones de tipos utilizados. No obstante, si el programa es suficientemente pequeño puede constar de un único módulo, el principal.

2.4.1.3 Ficheros de cabecera

Para poder llamar a una función desde otra, su código debe hallarse en el mismo módulo antes de ser llamada. De no ser así, puede ocurrir:

- Que el código se halle en otro módulo,

- Que el código se halle posteriormente en el mismo módulo, o
- Que se trate de una función de librería cuyo código será enlazado posteriormente.

En estos casos, la función debe ser declarada como una variable más antes de ser llamada. La declaración consta exclusivamente de la cabecera de la función, también llamada prototipo. Aunque sólo es necesario indicar los tipos de los parámetros, opcionalmente y por claridad, se pueden añadir también los nombres:

tipo nombre_función (lista de tipos de todos los parámetros);

En el caso de que la función se halle en otro módulo, también podría añadirse el modificador `extern` como en el caso de las variables, pero aquí resulta superfluo e innecesario, pues no hay ambigüedad posible entre lo que es la definición de la función (que incluye su código) y lo que es una simple declaración (sólo la cabecera, terminando la declaración con punto y coma).

Para facilitar la tarea de programación y evitar tener que declarar funciones a menudo, lo que puede llevar consigo errores, habitualmente por cada fichero fuente conteniendo código de funciones (con extensión .c) existe un fichero con el mismo nombre (y extensión .h) que contiene exclusivamente las cabeceras o prototipos (declaraciones) de las funciones.

De esta forma, basta con incluir el fichero mediante una directiva `#include` para declarar todas las funciones contenidas en `fichero.c`, pudiendo ser utilizadas sin más. De la misma forma, si se desea usar una función de librería, debe incluirse el fichero de cabecera que contiene la declaración de dicha función.

En el caso de la librería estándar, al hallarse estos ficheros en el directorio por defecto debe usarse la forma `#include <fichero.h>`

La directiva `#include` puede utilizarse también para incluir cualquier otro fichero que se desee como habitualmente suele ocurrir con ficheros que contienen definiciones de tipos, constantes, macros, etc.

2.4.1.4 Librerías

Las librerías constituyen una forma simple de reunir varios ficheros objeto conjuntamente. Las librerías pueden ser de dos tipos:

- **Estáticas**: El código de la función es integrado con el ejecutable en la fase de enlazado.
- **Dinámicas**: El código de la función es cargado cuando se ejecuta el programa. Las librerías dinámicas permiten economizar espacio en disco pero, sobre todo, memoria porque son cargadas sólo una vez en memoria y el código puede ser compartido entre todos los programas que la necesiten.

Las cabeceras o prototipos (declaraciones) de las funciones de cada librería se hallan en ficheros de cabecera con extensión `.h`. Tantos estos ficheros como las librerías se hallan en determinados directorios conocidos por el compilador.

2.4.1.5 Variables

Toda variable debe ser declarada antes de su utilización. No obstante, si bien el inicio de su ámbito de aplicación queda determinado por la declaración, la finalización de éste se determina atendiendo a las denominadas **reglas de ámbito**.

Según las reglas de ámbito, **C** distingue tres tipos de variables: locales, globales y parámetros formales.

- **Variables Locales**

Las variables definidas dentro de un bloque (delimitado por "{" y "}") serán locales, es decir, su ámbito se restringe exclusivamente a dicho bloque, que puede ser una función o una sentencia compuesta. A su vez, las variables locales pueden ser de tres tipos:

Automáticas: Se crean al entrar en el bloque y se destruyen al salir de él. Se definen anteponiendo la palabra clave `auto` al tipo de la variable en la declaración. Por defecto, todas las variables locales a un bloque son automáticas.

Estáticas: No son destruidas al salir del bloque, y su valor permanece inalterable cuando se vuelve a él. Se definen anteponiendo la palabra clave `static` al tipo de la variable en la declaración.

Registro: Son variables estáticas, exclusivamente de tipo `char` o `int`, que se almacenan en un registro de la CPU en vez de en la memoria.

Si se utiliza con variables referenciadas con mucha frecuencia se acelera la ejecución. Se definen anteponiendo la palabra clave `register` en la declaración.

- **Variables Globales**

Las variables globales son aquellas que se definen en un módulo (fichero fuente) fuera de cualquier función (incluida también la función `main`). Las variables globales existen siempre, y no pueden usarse registros.

Si bien el ámbito de una variable global se restringe al módulo en que es definida, este ámbito puede ampliarse a cualquier módulo que la declare, anteponiendo la palabra clave `extern`, que indica al compilador que su definición se halla en otro módulo. Será el enlazador quien se encargue de editar los enlaces adecuadamente, es decir, hacer corresponder ambas variables con la misma dirección de memoria.

No obstante, puede evitarse que una variable global sea exportada fuera del módulo donde es definida (es decir, sea visible desde otros módulos). Para ello debe anteponerse la palabra `static` en su definición, creando así una variable global estática. De esta forma, si en un módulo existe la siguiente definición

```
static int a;
```

la siguiente declaración en otro módulo

```
extern int a;
```

Provocará un error de enlazado, de la misma forma que si no se hubiese definido la variable original a importar.

En cambio, la definición

```
int a;
```

Se referirá a otra variable global distinta a la del primer módulo.

Aunque no es realmente necesario anteponer la palabra `extern` para importar una variable global no estática de otro módulo (el enlazador las considerará la misma variable), se recomienda hacerlo, pues ello ayudará a detectar errores de programación.

Por razones obvias, una declaración mediante `extern` no admite inicializaciones. Éstas sólo pueden ser realizadas en la definición original de la variable.

- **Parámetros Formales**

Los parámetros formales de una función son, a todos los efectos, variables locales automáticas, y su valor al entrar en la función es el resultado de asignarle el valor actual del argumento pasado en la llamada a ésta.

En **C** todos los argumentos pasados a una función son por valor. Es decir, cuando el argumento es una variable, tan sólo sirve para inicializar el parámetro formal con el valor actual de esta variable. Así, aunque el parámetro formal puede ser alterado posteriormente dentro de la función, como cualquier otra variable local, la variable pasada como argumento no puede ser modificada.

2.4.2 FIRMWARE DIFIJO

El primer diagrama de flujo para analizar es el general (figura 2.44), este comienza con la inicialización de los módulos internos del microcontrolador que nos es otra cosa que la habilitación y configuración de los periféricos como los temporizadores (Timer 0), interrupciones, etc., seguido por la inicialización del módulo USB del PIC18F4550, y la lectura de la memoria EEPROM interna, donde está las tarifas para los diferentes tipos de vehículos (Liviano, pesado, muy pesado y especial) y la forma de cobro (manual y automática). Si el cobro es automático se inicializa el módulo RTCC, para finalmente entrar en un lazo infinito.

La primera acción dentro del lazo infinito es chequear si existe conexión entre el dispositivo y el computador. Luego de esto se realiza las tareas propias del dispositivo que más adelante se detallan para finalmente regresar al inicio del lazo.

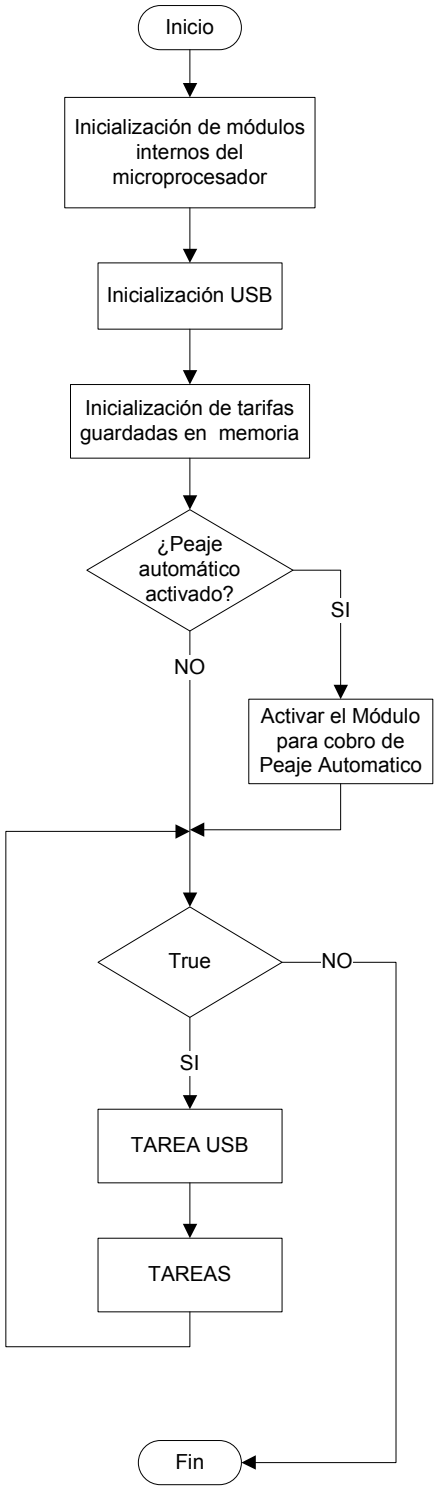


Figura 2.44 Diagrama de flujo general

El segundo diagrama de flujo (TAREAS), como se muestra en la figura 2.45 es donde se realizan las tareas que cumple el microcontrolador.

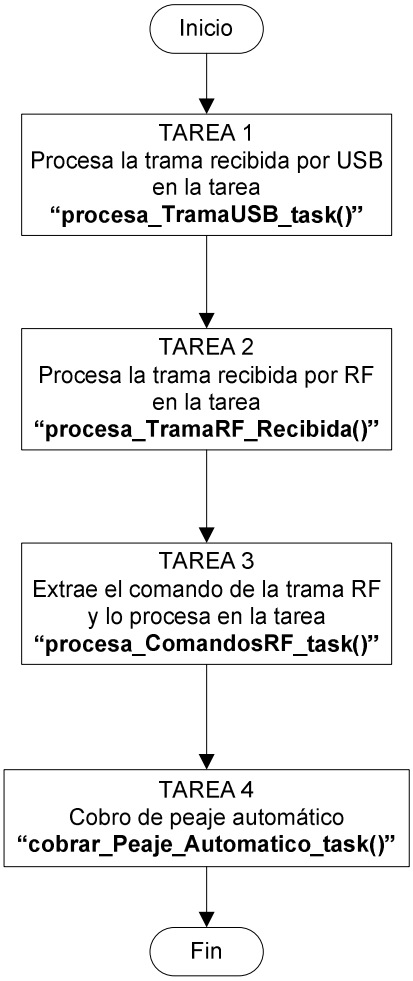


Figura 2.45 Diagrama de flujo de TAREAS

La función TAREA USB que se ejecuta antes de TAREAS no se detalla debido a que es un proceso propio del compilador utilizado (CCS). A continuación se detalla cada una de las tareas realizadas por la unidad micro-controladora con su respectivo diagrama de flujo.

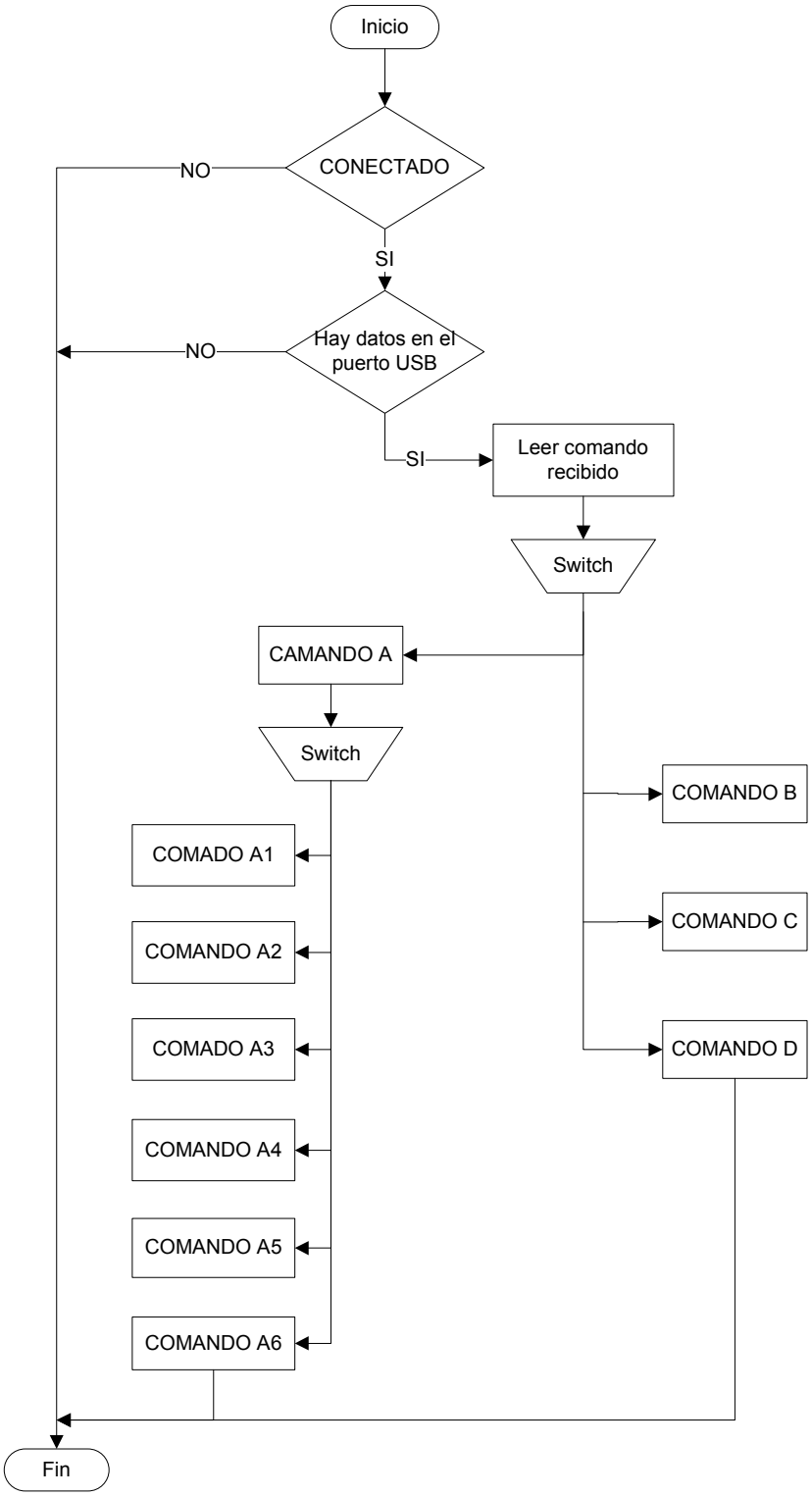


Figura 2.46 Diagrama de flujo de TAREA 1

TAREA 1 (figura 2.46) se encarga de la transferencia de datos por el bus USB, inicia preguntando si existe una conexión en el bus, esto lo realiza la función interna del compilador ***usb_cdc_conected()***, luego de verificar la conexión, revisa si hay datos en el puerto.

Los datos que recibe forman una trama y dentro de esta trama se encuentra el campo "comando" que determina el tipo de trama que enviará ya sea por el mismo puerto USB o por el puerto serial (radiofrecuencia) hacia el DIMOVIL. En la tabla 2.6 se describe cada comando. Dentro del comando solicitudes hay varias opciones que se detallan en la tabla 2.7.

COMANDO	DESCRIPCIÓN
Comando A	Solicitudes
Comando B	Recarga de saldo al vehículo
Comando C	Graba en la memoria nuevas tarifas
Comando D	Activa/Desactiva la forma de cobro (manual o automática)

Tabla 2.6 Comandos Trama USB

COMANDO	DESCRIPCIÓN SOLICITUDES
Comando A1	ID Vehículo
Comando A2	Saldo del vehículo
Comando A3	Tarifas que se están cobrando
Comando A4	ID del DIFIJO (Caseta)
Comando A5	Estado de la forma de cobro (Manual o Automática)
Comando A6	Cobro del peaje

Tabla 2.7 Descripción de solicitudes

La figura 2.47 describe a TAREA 2, en esta tarea se chequea si hay datos en el puerto serie. Si es así el microcontrolador compara los primeros datos recibidos con el inicio de la trama, si coinciden, lee los demás campos de la trama.

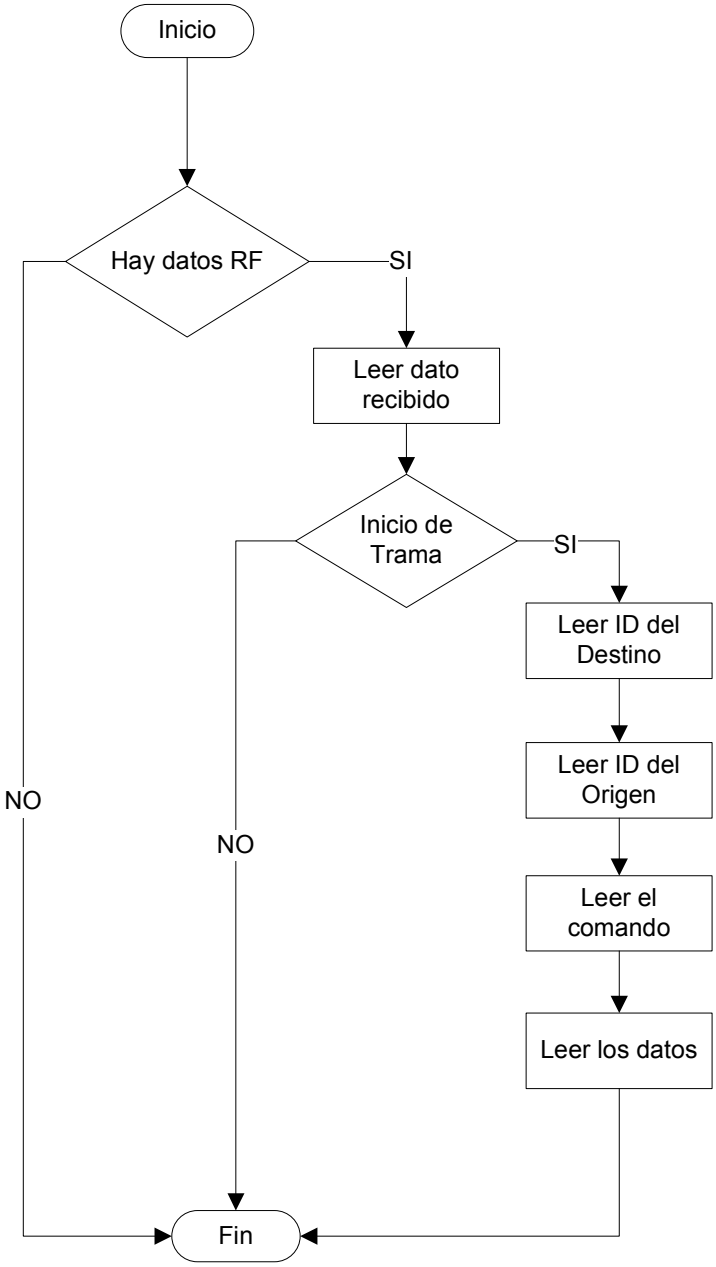


Figura 2.47 Diagrama de flujo de TAREA 2

TAREA 3 (figura 2.48) es la encargada de extraer el campo comando de la trama recibida por el puerto serie (DIMOVIL). Para este caso se tiene dos opciones, que el DIMOVIL responda su ID (Placa) o el saldo que tiene. Esos datos se envían por el puerto USB para ser visualizado en la interfaz de usuario.

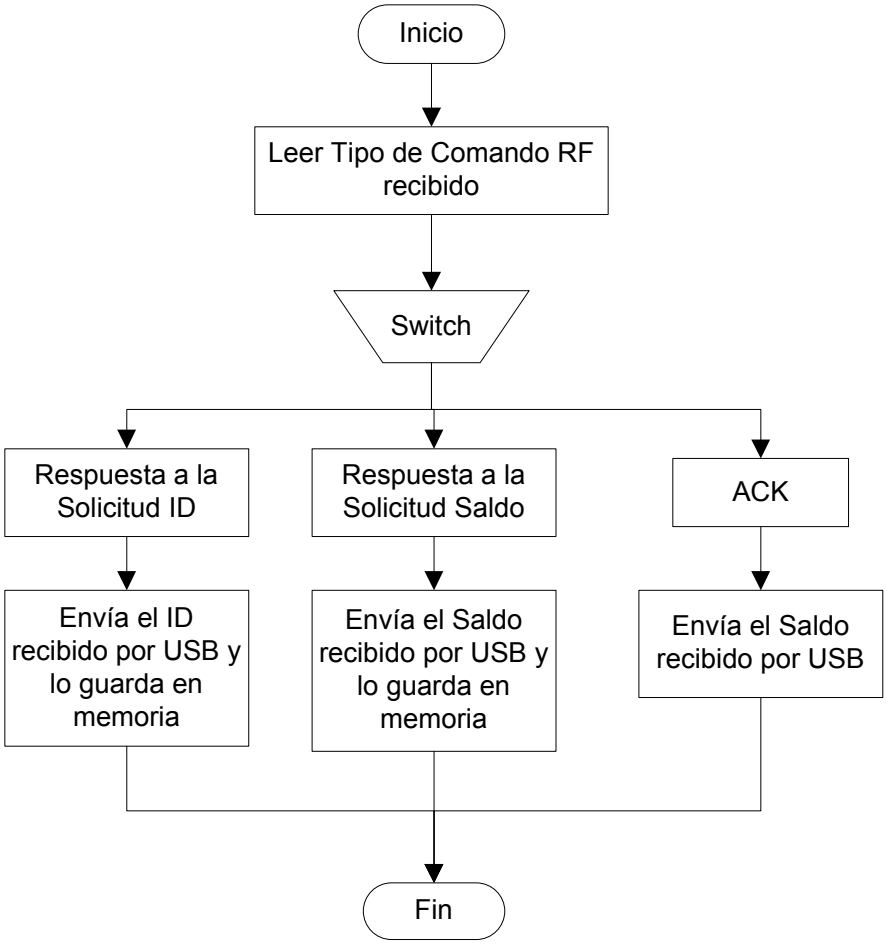


Figura 2.48 Diagrama de flujo de TAREA 3

TAREA 4 (Figura 2.49) se encarga finalmente se realizar el cobro del peaje. Para dicho propósito se inicia preguntando la forma de cobro. Si el cobro es automático, envía una solicitud de ID, cuando recibe el ID del auto se procede con el cobro del peaje.

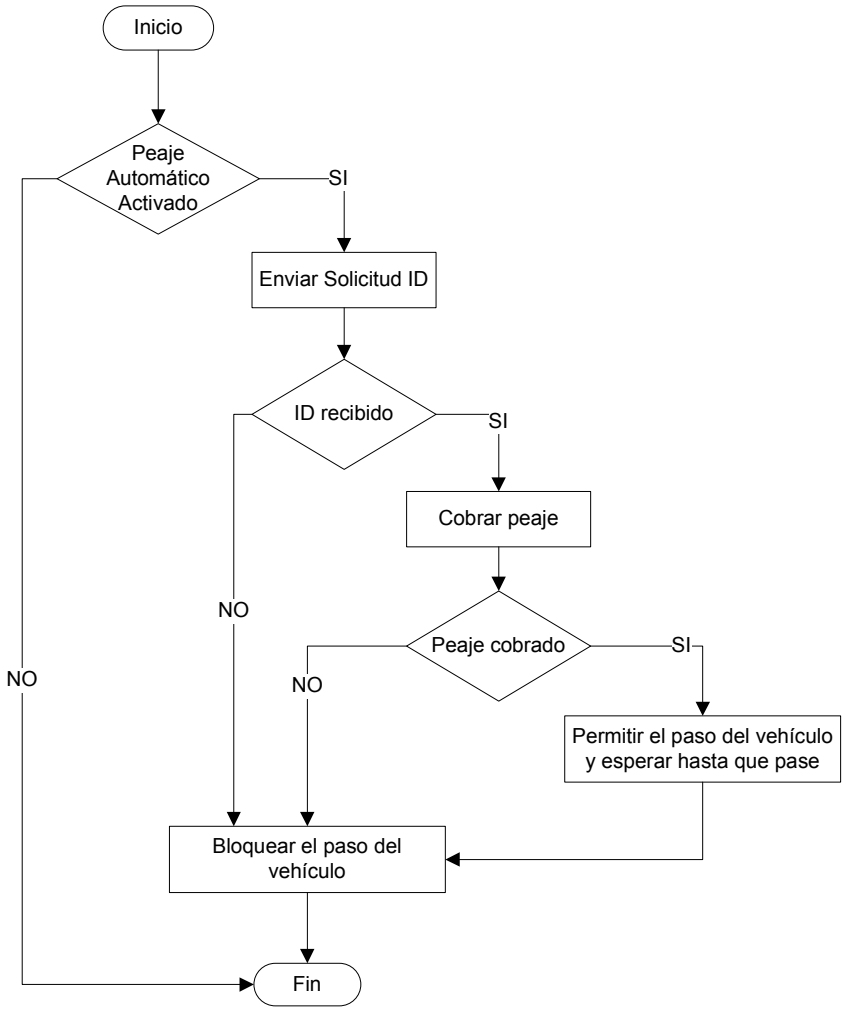


Figura 2.49 Diagrama de flujo de TAREA 4

El Firmware se encuentra en el ANEXO A.

2.4.3 FIRMWARE DIMOVIL

El diagrama de flujo general para el DIMOVIL (figura 2.50) comienza con la inicialización de los módulos internos del microcontrolador, seguido de la lectura del saldo actual que dispone, el cual se encuentra guardado en la memoria externa (24LC08B), según los valores del saldo disponible, actualiza el estado de los LEDs, para finalmente entrar a un lazo infinito.

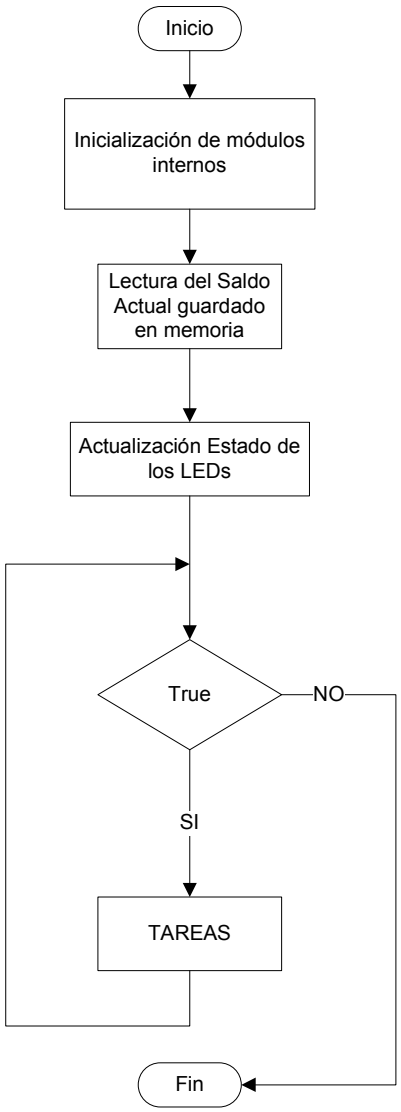


Figura 2.50 Diagrama de flujo general (DIMOVIL)

En el lazo infinito el microcontrolador está chequeando si existe datos en el puerto serie (datos que recibe por RF del DIFIJO). Luego de esto realiza las tareas propias del dispositivo para finalmente regresar al inicio del lazo.

La figura 2.51 muestra las tareas que se realizan dentro del lazo infinito.

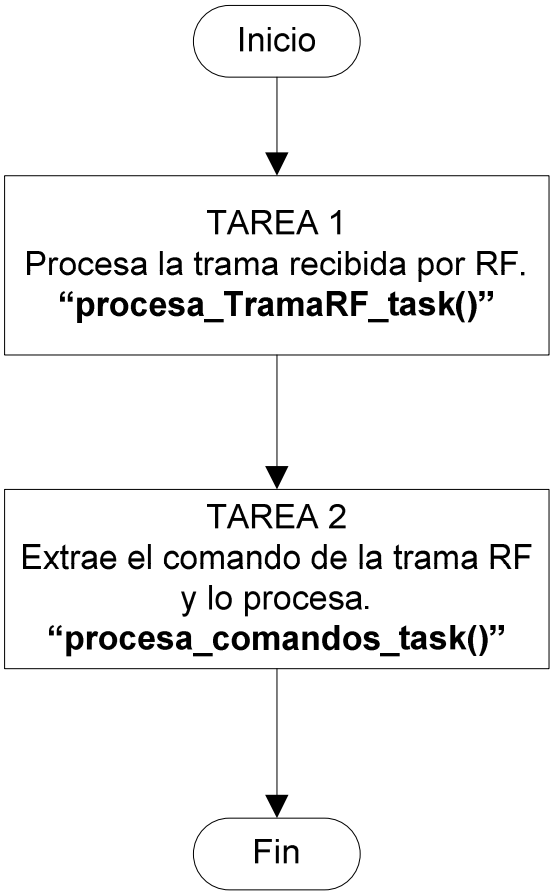


Figura 2.51 Diagrama de flujo de TAREAS (DIMOVIL)

La TAREA 1 (figura 2.52) es la encargada de recibir todos los datos válidos del puerto serie y leer los diferentes campos de la misma. Los primeros datos que llegan los compara con los bits del inicio de trama previamente definidos, si está correcto lee los otros datos.

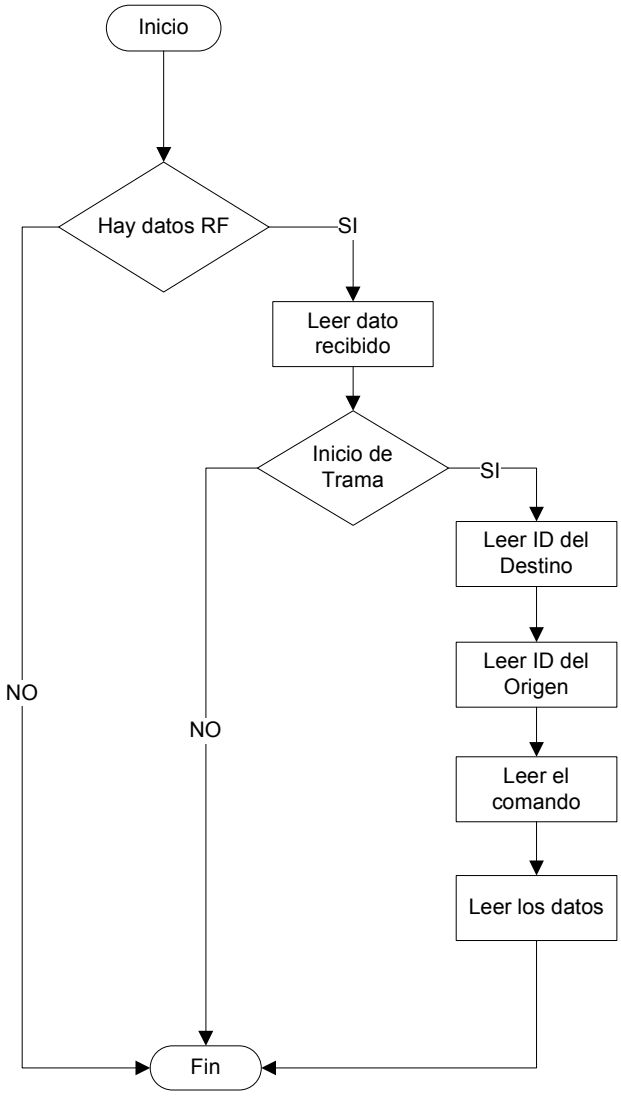


Figura 2.52 Diagrama de flujo de TAREA 1 (DIMOVIL)

La TAREA 2 (figura 2.53) procesa el campo comando de la trama recibida. Con este campo se proporciona información al DIFIJO como el ID del auto (Placa) y el saldo que dispone o realiza modificaciones en la memoria externa como recargar el saldo o descontar el valor del peaje.

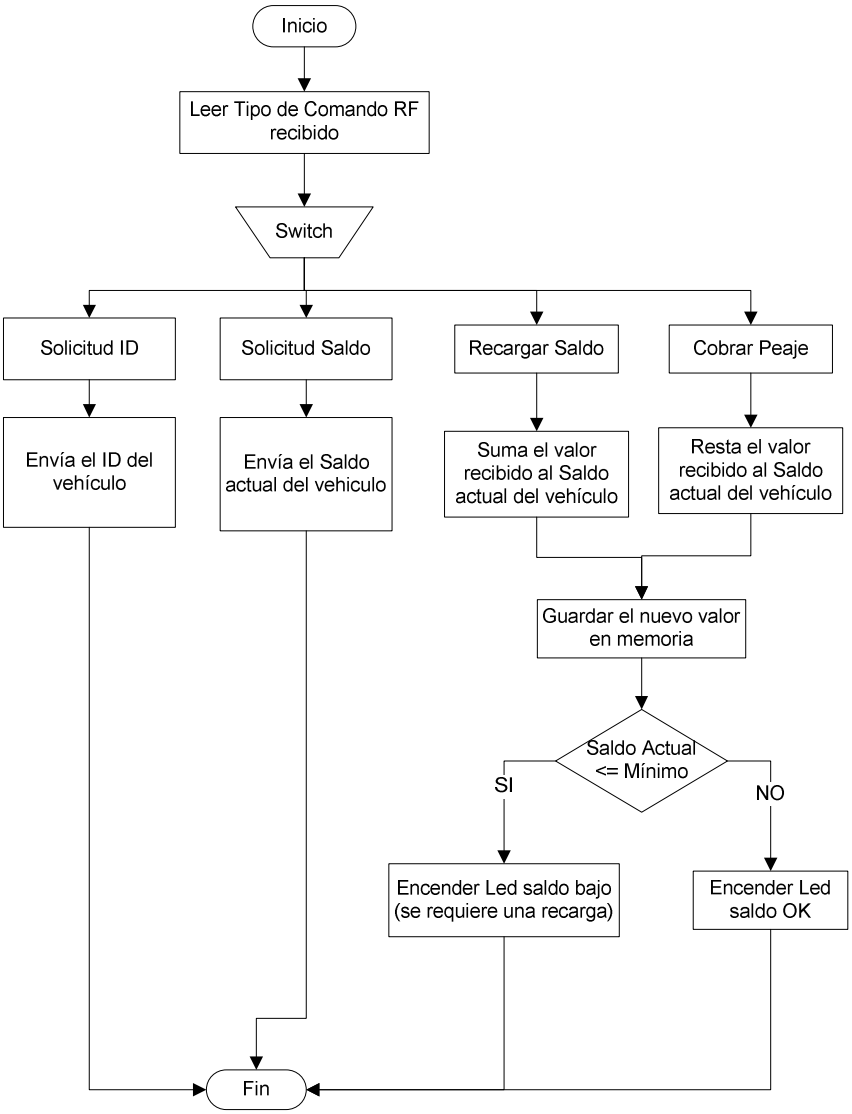


Figura 2.53 Diagrama de flujo de TAREA 2 (DIMOVIL)

El firmware se encuentra en el ANEXO B

2.4.4 INTERFAZ DE USUARIO

La conexión entre la PC y el DIFIJO se realiza con un cable USB. Esta conexión emula una comunicación serial, entonces, cuando se conecta el dispositivo se debe chequear que puerto **com** se crea, esto se revisa en la ventana de administración de dispositivos, la cual se aprecia en la figura 2.54, en este caso se crea el puerto **com4.**

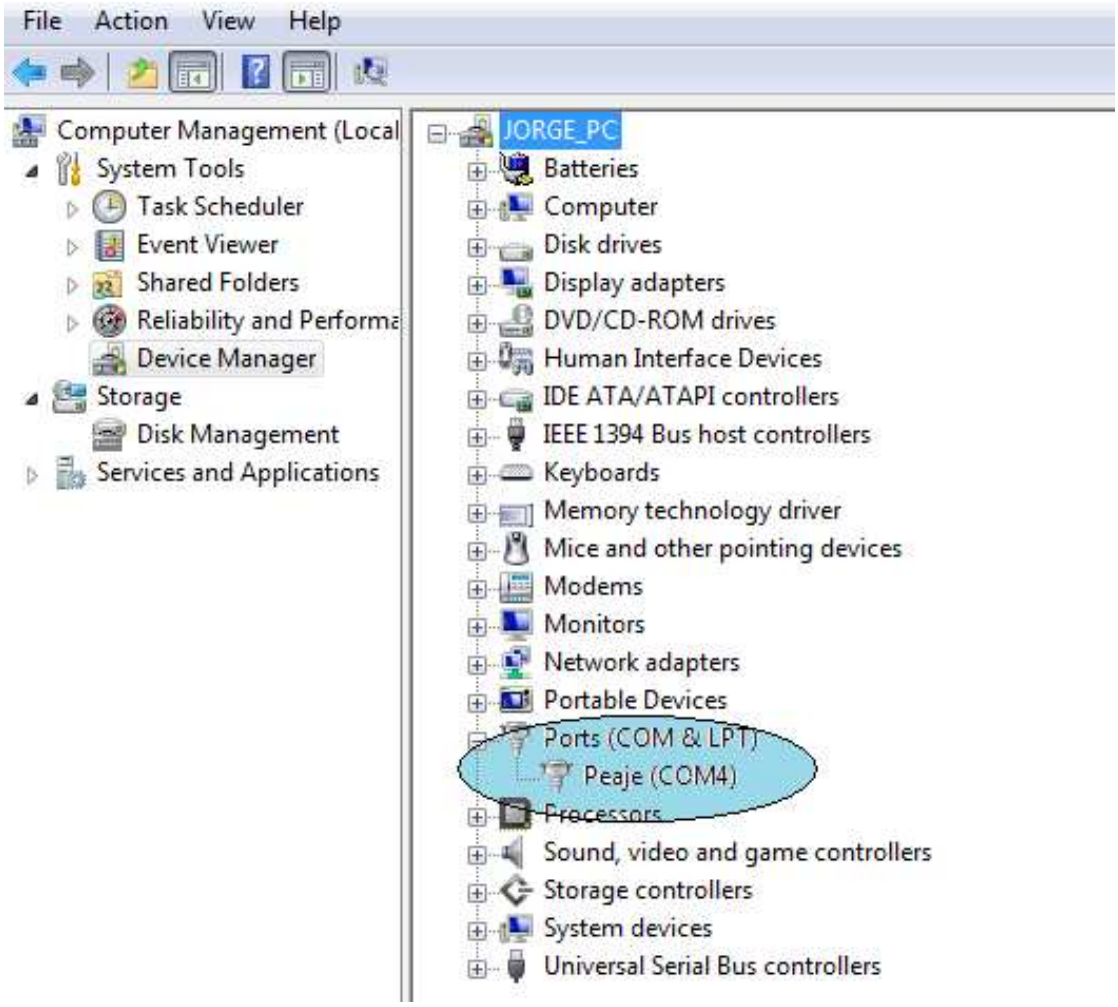


Figura 2.54 Ventana de Administración de Dispositivos

El programa “interfaz de usuario” es desarrollado en el lenguaje de programación visual C#.

Para crear un proyecto se inicia el programa Microsoft Visual Studio escogemos: Archivo → Nuevo → Proyecto.

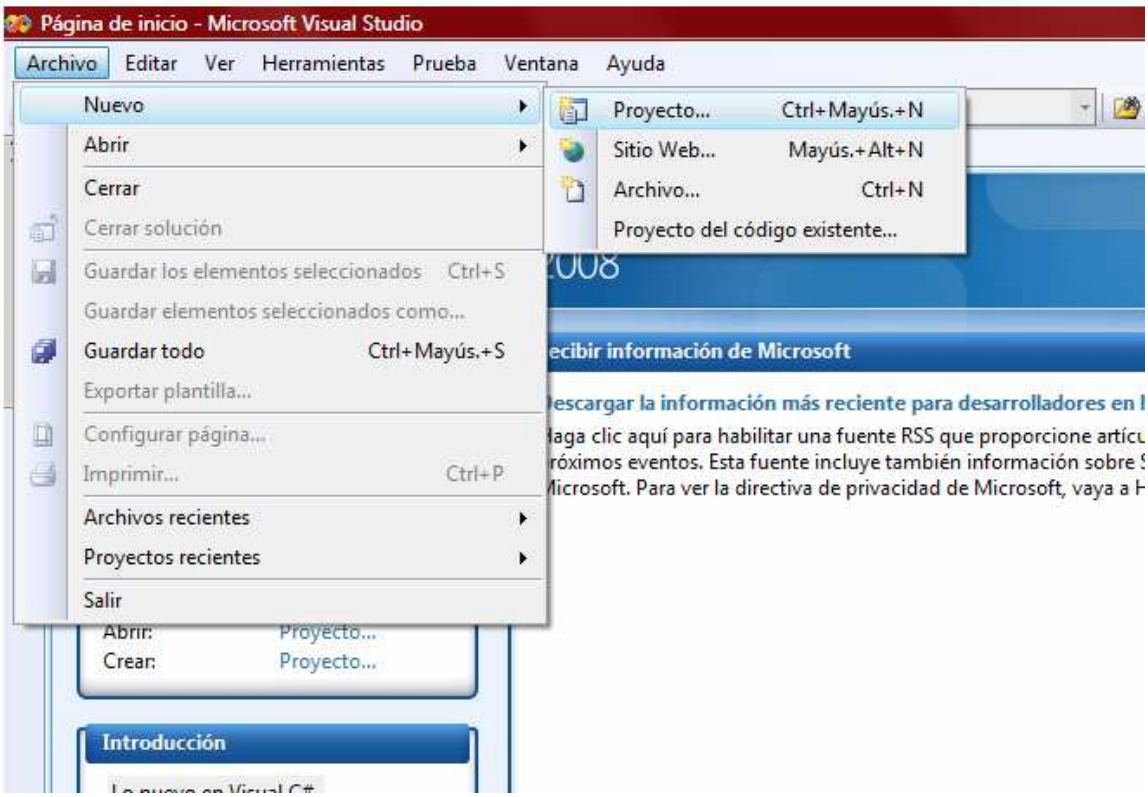


Figura 2.55 Crear un Programa

En la ventana que aparece se escoge “Aplicación de Windows Forms”, se nombra al proyecto y se selecciona donde se va a grabar, como se muestra en la figura 2.56.

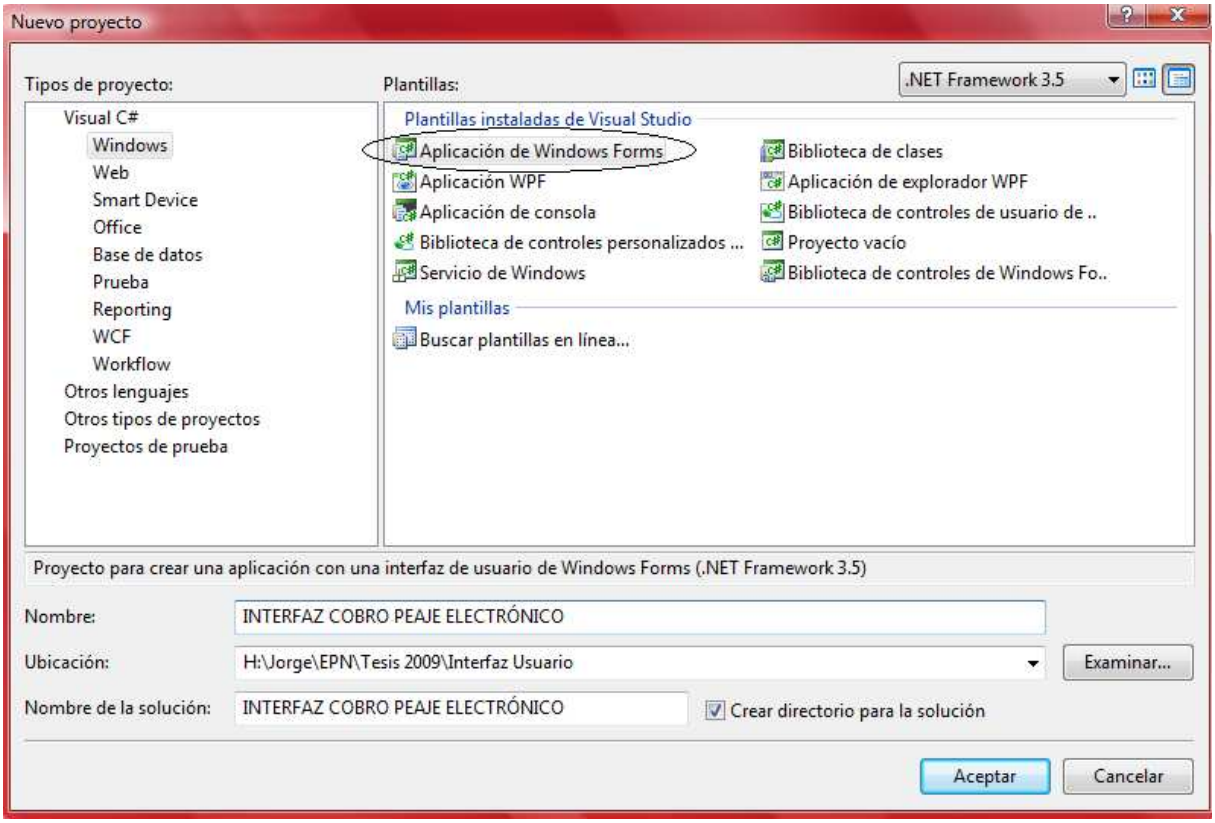


Figura 2.56 Nuevo Proyecto

Una vez realizado estos procesos se tiene lista la pantalla para desarrollar el proyecto. Como se puede ver en la figura 2.57, se ha creado una ventana llamada “Form1” en la cual se va a diseñar el formulario.

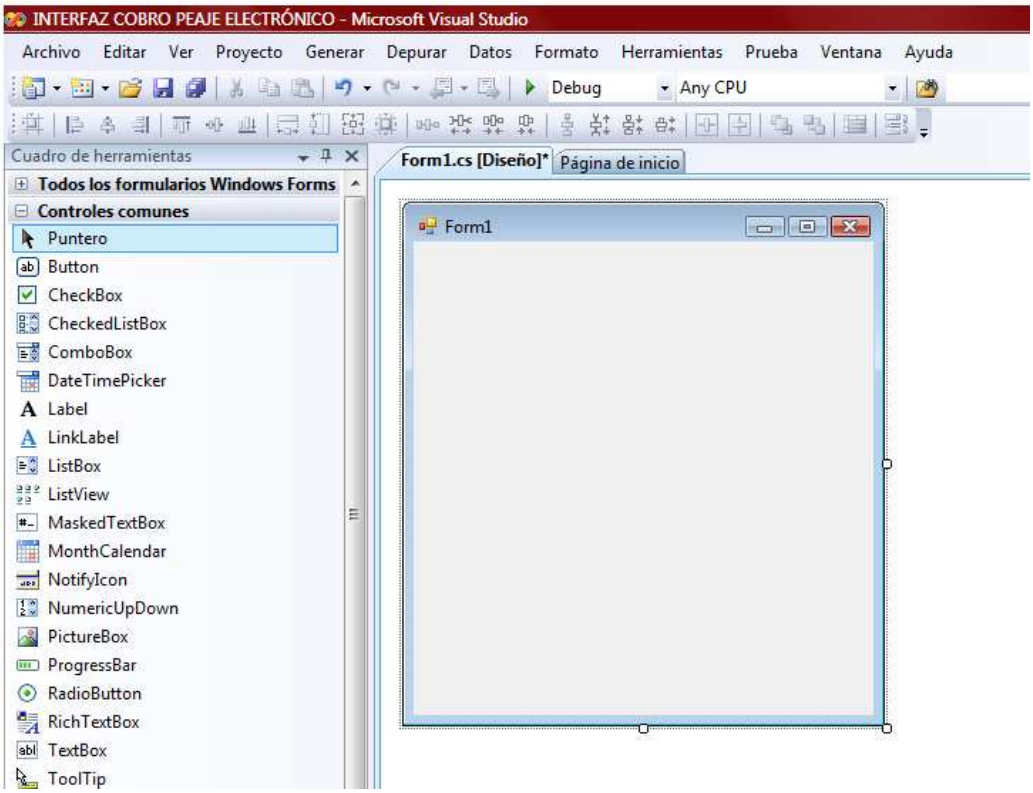


Figura 2.57 Pantalla Para el Desarrollo del Programa

Para guardar el proyecto hay que pulsar el botón “Guardar todo” como se muestra en la figura 2.58.

Figura 2.58 Guardar un Proyecto

La aplicación desarrollada está compuesta por una pantalla principal que contiene cuatro menús (Figura 2.59)

Figura 2.59 Pantalla Principal

A continuación se detalla los eventos de la interfaz gráfica de la aplicación.

EnviarTramaUSB (TipoTrama, Parametros). Esta función se encarga de enviar una trama hacia el DIFIJO usando la comunicación USB, se envía como parámetros el tipo de trama y los datos que contendrá la misma.

serialPort1_DataReceived. Este es un evento asociado a la emulación del puerto serial y se ejecuta cada vez que llega un carácter al búfer de recepción.

timer1_Tick. Evento que se ejecuta cada 200 milisegundos, es el encargado de procesar la información que llega al computador en forma de trama desde el DIFIJO, está en constante ejecución debido a la comunicación asincrónica.

Form1_Load. Se ejecuta cuando se carga la aplicación en el sistema operativo de la computadora, este punto es ideal para escanear los puertos seriales disponibles en el computador.

LeerTarifas_Click. Evento asociado a un botón del interfaz el que permite leer las tarifas almacenadas en el DIFIJO

CargarSaldo_Click. Evento asociado a un botón del interfaz que envía un valor en dólares para que sea añadido al saldo de un DIMOVIL. Este botón estará activo si DIFIJO tiene un DIMOVIL disponible.

CargarNuevasTarifas_Click. Actualiza las tarifas que se cobran por concepto de peaje. Esta información está guardada en el DIFIJO.

BuscarVehiculo_Click. Escanea si algún DIMOVIL está acercándose al peaje, solicitándole que se identifique.

conectarEquipo_Click. Abre la conexión al puerto serial y verifica si existe un DIFIJO conectado.

cobroPeajeManual_Click. Realiza el cobro de peaje de forma manual, este botón esta activo si algún DIMOVIL está al alcance.

LeerSaldo_Click. Lee el saldo que tiene un DIMOVIL

timerTimeout_Tick. Este evento se encuentra asociado al tiempo máximo que puede esperar una respuesta del DIMOVIL. Cuando se ejecuta este evento deshabilita todos los botones y es necesario escanear en busca de un DIMOVIL.

botonEstadoPeaje_Click. Se consulta el estado de cobro: automático o manual que se encuentra guardado en el DIFIJO.

DesactivarBotones. Desactiva todos los botones que tengan que ver con la comunicación hacia EL DIMOVIL

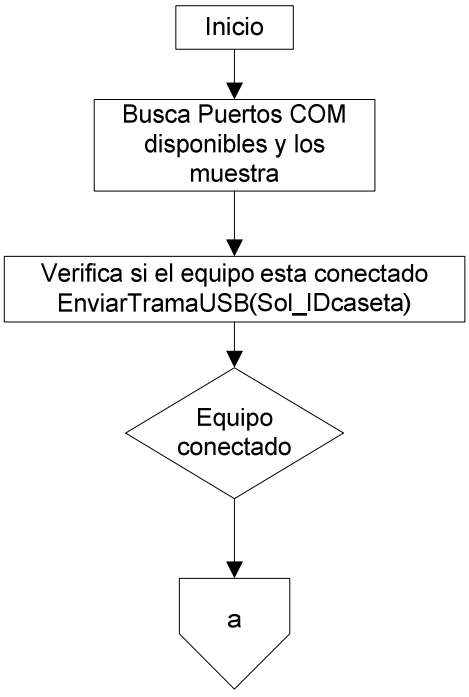
ActivarBotones. Activa todos los botones que tengan que ver con la comunicación hacia EL DIMOVIL.

serialPort1.Write (Datos). Envía caracteres por el puerto serial, esta es una función propia del lenguaje de programación.

serialPort1.ReadExisting(). Lee todos los caracteres que están en el búfer de recepción serial, esta es una función propia del lenguaje de programación.

El diagrama de flujo de la figura 2.60 corresponde al programa para la interfaz de usuario. El código fuente se encuentra en el ANEXO E. En el ANEXO F se encuentra el manual de la “INTERFAZ TELEPEAJE”.

Diagrama Flujo General Software PC



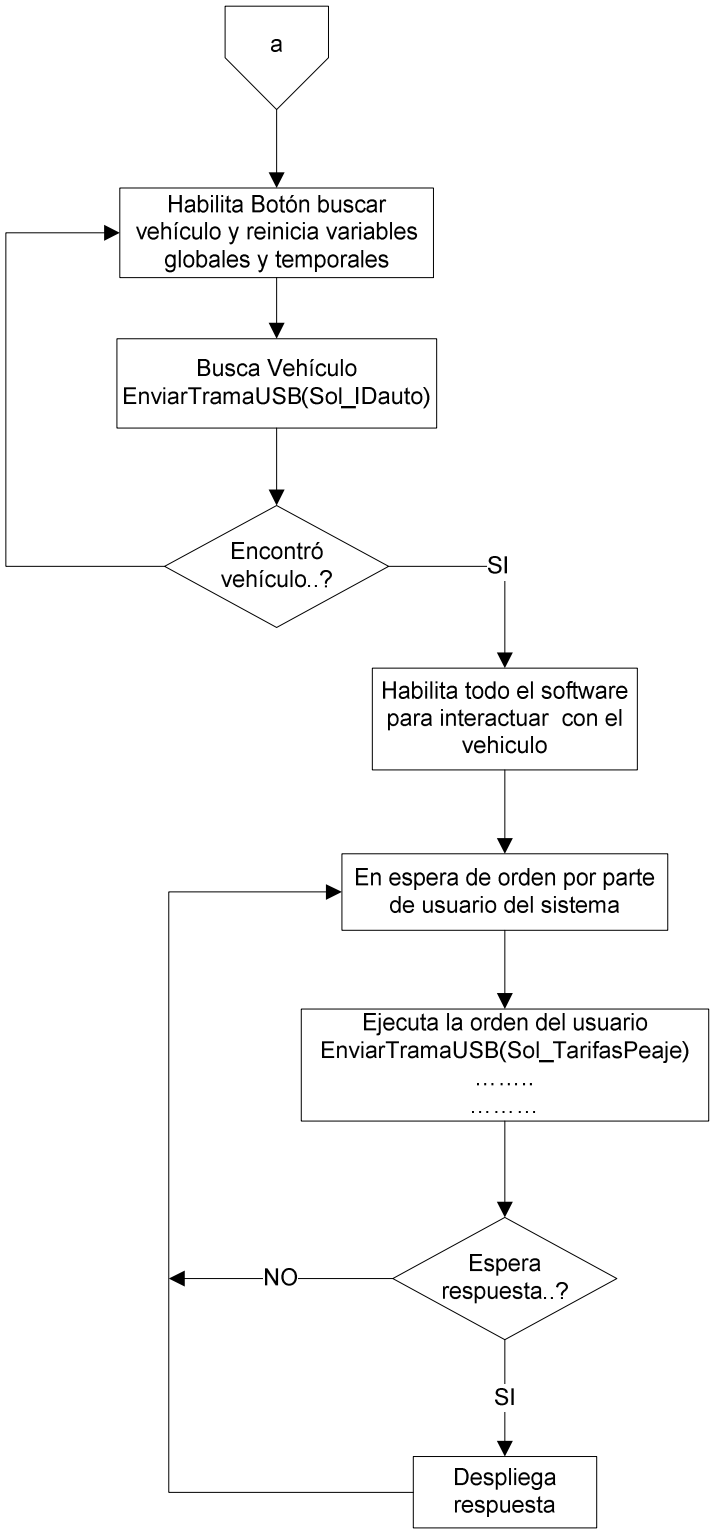


Figura 2.60 Diagrama de flujo para la PC

2.5 SEGURIDADES DEL SISTEMA DE COBRO

Con el bloque de programación In Circuit que tienen los dispositivos se puede copiar el código fuente del microcontrolador y realizar cambios, por ejemplo la placa del auto y el saldo disponible, para evitar estos inconvenientes protegemos el código de cualquier lectura y eso se logra utilizando las directivas fuses.

Esta directiva define que fusibles deben activarse en el dispositivo cuando se programe, para el caso de protección de lectura se utiliza:

```
#fuses PROTECT
```

CAPÍTULO 3

3. APLICACIÓN DEL PAGO AUTOMÁTICO DE PEAJES

3.1 INTRODUCCIÓN[67]

Esta tecnología se desarrolló en los años ochenta y se lo llama "telepeaje" o "cobro electrónico de peajes", con lo cual los usuarios pueden pasar por las estaciones sin parar o reduciendo un poco la velocidad.

El País pionero en el uso generalizado del telepeaje ha sido Noruega, el primer uso exitoso fue en la ciudad de Bergen en 1986, cuando entró en funcionamiento como complemento al cobro tradicional con casetas de cobro de peaje.

Los sistemas 100% electrónicos, sin estaciones de peaje se han usado en varios países como: Canadá desde 1997, en varias carreteras de Noruega, Australia, Argentina, Israel y en Santiago de Chile, siendo el sistema chileno el primero del mundo que atraviesa el centro comercial de la ciudad, en un tramo de una autopista urbana congestionada y opera con los vehículos transitando a velocidad normal.

Los telepeajes son conocidos mundialmente como ETC (Electronic Toll Collection), los beneficios de la aplicación de esta tecnología de pago de peaje son:

- No se generan colas de espera por el pago de peaje
- Aumenta la seguridad y comodidad del viajero al no utilizar dinero en efectivo
- Aumenta la seguridad y disminuyen los fraudes por el operador de la carretera

[67] http://es.wikipedia.org/wiki/Cobro_electr%C3%B3nico_de_peajes

- Aumenta la flexibilidad para definir tarifas: variantes en el tiempo, o dependiendo de condiciones o estados de operación del sistema
- Disminuye costos por administración y gestión del dinero (Transporte de monedas y billetes, depósitos, giros, etc)

Figura 3.1 Telepeaje 100% Electrónico: Autopista Costanera Norte Santiago - Chile[68]

3.2 TECNOLOGÍA UTILIZADA EN LOS PEAJES

El sistema se basa en el uso de la identificación vehicular por medio de radiofrecuencia (RF). Para ello se utilizan dispositivos electrónicos en los vehículos llamados TAGs o Transponders.

[68] http://es.wikipedia.org/wiki/Autopista_Costanera_Norte_(Santiago_de_Chile)

El sistema electrónico de peajes se estructura de la siguiente manera:

- Sistemas de vía o puntos de cobro
- Sistema central de gestión
- Sistema de comunicaciones

Sistemas de vía o puntos de cobro

El equipamiento de los sistemas de vía se instala sobre pórticos, que son estructuras en forma de arcos por sobre las vías, cubriendo las pistas y la berma[69] de la carretera. Tiene los siguientes subsistemas:

- Identificación automática de vehículos (AVI)
- Clasificación automática de vehículos (AVC)
- Sistema de fiscalización por video (VES)

El sistema AVI se basa en la detección e identificación del TAG instalado en el vehículo a través de lectores y antenas ubicadas en el pórtico, sobre las pistas.

Las tecnologías AVI aplicada al cobro de peaje se han basado y se basan actualmente en: 1) la emisión e intercepción de señales electromagnéticas provenientes del vehículo con la información que contiene la identificación, 2) la recuperación de la información contenida en esta señal y 3) procesamiento de la información para relacionar la identificación del vehículo:

[69] Zona lateral adyacente a la calzada de la vía

Las tecnologías aplicadas para este fin son:

- ✓ **Sistemas de lazos inductivos**. Este sistema utiliza un lazo como antena enterrada en el pavimento de las pistas de la carretera para comunicarse con un TAG instalado en el vehículo. Utiliza rango de frecuencias muy bajos. Fue la primera tecnología utilizada en este tipo de aplicación.
- ✓ **Sistemas ópticos basados en video.** Este sistema está basado en el uso de cámaras de video que capturan la placa patente de los vehículos y procesan la imagen capturada con el objeto la identificación de la placa del patente.
- ✓ **Sistemas ópticos basados en código de barras.** Estos sistemas se basan en la lectura a través de un lector láser de un código de barras adherido al vehículo (sticker).
- ✓ **Sistema de Radiofrecuencia (RF) en el rango de microondas.** Es el sistema más utilizado en el mundo. Se basa en un dispositivo llamado TAG o Transponders instalado en el vehículo y que emite ondas RF con la información del vehículo, la cual es captado por lectores y antenas ubicadas sobre las pistas de las carreteras. Básicamente existe 2 tipos de TAGs: activos y pasivos.

El sistema AVC clasifica el vehículo, es decir determina si se trata de un vehículo liviano (Auto o camioneta), vehículo pesado (bus o camión), motocicleta y vehículos con acoplado[70]. Existen varias tecnologías para realizar esta labor, como por ejemplo por imágenes de video, por rayos infrarrojos, por rayos láser y por peso, entre otras.

[70] Motoniveladora, Bulldozer, Retroexcavadora de banda, Pala, Terminadora Asfáltica, Chata para transporte de maquinaria

TECNOLOGÍA	VENTAJAS	DESVENTAJAS
Lazos Inductivos	La cercanía que existe entre el TAG y la antena aumenta la confiabilidad. Muy bajo potencial de interferencias eléctricas. Bajo potencial de interferencia con pistas adyacentes. Los lazos pueden ser utilizados para otras aplicaciones de control de tráfico.	Bajas tasas de transmisión de datos. Dificultad media en la duplicación de TAGs. Normalmente los TAGs requieren de energía desde el vehículo. Instalación del TAG en el vehículo comparativamente más complicada. sensible a condiciones ambientales
Ópticos basados en video	No se requiere de TAG. No hay posibilidad de interferencia entre pistas.	Procesamiento de imágenes altamente intensivo en recursos computacionales. Sensible a fallas si la placa del vehículo está sucia o en mal estado. Sensible a condiciones ambientales como niebla, lluvia y nieve. Altamente demandante de Ancho de Banda para los sistemas de comunicaciones.
Ópticos basados en código barras	Simplicidad del TAG en el vehículo al tratarse de un adhesivo con un código de barras impreso. Baja posibilidad de interferencia entre pistas. Lectura rápida del código.	Los TAGs son fáciles de duplicar. Susceptible a fallas por lluvia, nieblas, suciedad o humedad en el TAG. Fuertes restricciones en la posición y velocidad del vehículo que pasa frente al lector.
RF / Microondas Activos	Mayor rango de operación que el TAG pasivo porque el TAG no es energizado por la onda interrogadora. Mayor confiabilidad que el TAG pasivo porque la respuesta de TAG es más potente. Menor posibilidad de interferencia eléctrica.	TAG más caro que el pasivo, pues es de mayor complejidad. Alta probabilidad de interferencia entre pistas por la potencia de las señales. El TAG requiere de una batería o conectarse a una fuente de poder el vehículo.
RF / Microondas Pasivos	El TAG no requiere de batería ni estar conectado al vehículo. El TAG es más simple que el caso activo.	Menor confiabilidad que un TAG activo. Más susceptible a interferencia eléctrica.

Tabla 3.1 Comparación de los diferentes medios de pago[71]

[71] Fuente: U.S National Cooperative Highway research Program. Electronic Toll and Traffic Management (ETTM) systems

Los sistemas de VES capturan la imagen del vehículo, en particular de la placa patente, que han cometido alguna infracción, lo cual es detectado con la combinación y análisis adecuado de la información aportada por los sistemas AVI y VES y la información almacenada. Para realizar esta función, normalmente se utilizan cámaras de video digitales de alta velocidad y sistemas de iluminación (flash) infrarrojos. Las anomalías que el sistema VES trata de fiscalizar son: estado financiero del TAG (al día, moroso), TAG robado, TAG en mal estado, TAG no correspondiente al vehículo, entre otras.

El procesamiento de la información proveniente de los sistemas AVI, AVC más la información proveniente de la base de datos maestras y la coordinación y disparo es realizado por un procesador o controlador denominado “Controlador de pista”.

Normalmente existe uno de estos controladores por cada pista a supervisar. En cada punto de cobro existe otro procesador, denominado “servidor de punto de cobro”, el cual se conecta a los respectivos controladores de pista del punto de cobro en cuestión y también envía la información (transacciones e imágenes) al sistema de gestión central.

Sistema central de gestión

El sistema central de gestión está estructurado de los siguientes centros de información y de gestión:

- Centro de operaciones
- Centro de atención al cliente

El centro de operaciones se preocupa de la administración, supervisión, control y procesamiento de la información relacionada con los puntos de cobro. Las funciones principales que realizan son:

- Comunicación con los puntos de cobro (sistemas en vía)
- Administración de la transacciones
    - Reconocimiento automático de las placas patentes
    - Procesamiento manual de imágenes para interpretación de las placas patentes
    - Relación placa patente – TAG
    - Validación de situaciones dudosas
    - Validación de los problemas con las categorías o clases de los vehículos
    - Tarificación o valoración de las transacciones
- Administración de las tarifas
- Gestión de alarmas y telecomandos con los puntos de cobro
- Interfaces con sistemas externos

El centro de atención al cliente se preocupa de la relación del sistema de peaje electrónico con los usuarios de las vías. Sus funciones principales son:

- Administración de cuenta de clientes
    - Apertura de cuentas
    - Mantenimiento de cuentas

    - Cierre de cuentas
    - Marketing
    - Telemarketing
- Administración de TAGs
- Administración de avisos de cobranza
- Administración de infracciones
- Administración de la recaudación
- Generación de informes
- Administración de interfaces contable

Sistema de comunicaciones

Este sistema se encarga de realizar la transferencia de datos entre los puntos de cobro y el sistema de gestión central. Por lo general se implanta una red WAN de un gran Ancho de Banda, generalmente se utiliza Fibra Óptica como medio físico de transmisión.

TAGs o Transponders

Son dispositivos de identificación por RF, de tecnología y apariencia física similar a una tarjeta inteligente, es decir posee un microprocesador, una memoria y una antena. Los TAGs pueden ser de sólo lectura o también de escritura y lectura de su memoria. Poseen un identificador único, el que puede ser asociado al vehículo, o al conductor.

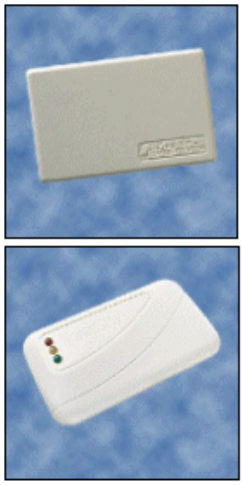


Figura 3.2 TAG o Transponders

La energía necesaria para establecer la comunicación con el lector la pueden obtener de una pila o batería (TAGs activos), o pueden ser inducidas por el lector a través de RF (TAGs pasivos).

Dependiendo del tipo de TAG, las velocidades máximas de lectura oscilan entre los 50 y 300 Km/h.

Existen numerosas estandarizaciones de los TAGs, especialmente del protocolo de comunicación entre el TAG y el lector. La estandarización de esta comunicación permite la independencia del fabricante de TAGs con el fabricante de lectores. En Chile, las carreteras más importantes están adoptando el estándar europeo CEN TC278 GSS1 A1

Costos

Los TAGs o Transponders utilizados en aplicaciones de peaje electrónico tienen un costo que varía entre unos US$ 7 a US$ 40 la unidad. Los TAGs compatibles CEN (norma europea) tienen un costo entre US$ 25 a US$ 28 la unidad, dependiendo del volumen, el fabricante y la funcionalidad específica.

Un proyecto de peaje electrónico para una determinada carretera, con un alto nivel de funcionalidad, con tecnología de última generación, performance altísimo, para flujos vehiculares altos, flujo libre en multipistas (no canalizados), puede llegar a costar del orden de US$ 700.000 a US$ 900.000 por pista.

3.3 TELEPEAJES: TUNEL GUAYASAMÍN

Actualmente en Pichincha está funcionando 2 telepeajes, uno en el túnel de la avenida Guayasamín que administra la EMOP, funcionando desde Noviembre/2006 y otro en la avenida Valles de Los Chillos que administra El Concejo Provincial.

Figura 3.3 Peaje Túnel Guayasamín

La implementación de la plataforma de estos 2 telepeajes la realizó la empresa PROCELEC, y esta es la que provee los TAG.

El sistema de cobro consta de los siguientes subsistemas:

- Sensor
- Radar
- Servidor (OPTO22 SNAP-PAC R1)
- Internet
- Centros de recarga

En la figura 3.4 se observa el sensor y radar de este sistema de cobro.

Figura 3.4 Sensor y Radar en el peaje Túnel Guayasamín

Cuando un auto se está acercando al telepeaje por el carril designado para este propósito, el sensor lo detecta, envía esta información a un servidor, este enciende un radar el cual se comunica con el TAG que se encuentra en el auto, el TAG envía un ID que es único para cada vehículo, el radar detecta esta información y lo envía al servidor para ser procesada.

En la figura 3.5 se muestra el diagrama general de funcionamiento de estos telepeajes.

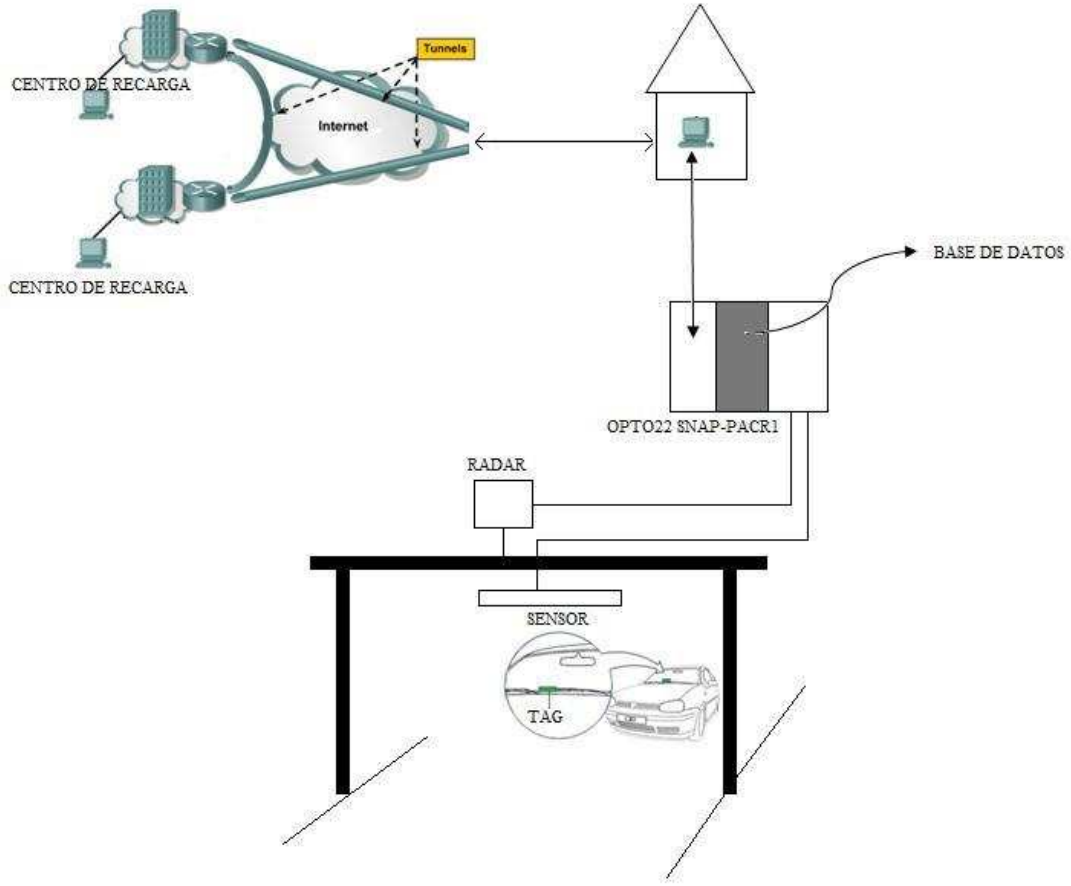


Figura 3.5 Diagrama Telepeajes Pichincha

El servidor OPTO22 se compone de 3 bloques:

- Interfaz de comunicación con el radar y el sensor
- Base de datos
- Interfaz de conexión con el internet

El servidor contiene en la base de datos información para cada número de TAG (Saldo que tiene, Tipo de Plan, Nombre del Usuario o número de cédula), Cuando el servidor recibe la información que le envía el radar, este lo procesa y graba en la base de datos.

Esta base está continuamente actualizándose con las recargas que realizan los usuarios. El servidor se comunica con las estaciones de recarga mediante una Red Privada Virtual (VPN: Virtual Private Network) a través del internet.

La figura 3.6 se observa los TAGs utilizados para estos peajes y en la figura 3.7 Y 3.8 se aprecia la interfaz de usuario que se utiliza para este sistema de cobro.

Figura 3.6 TAG de la Casa Fabricante NEDAP[72]

[72] Data Sheets: Nedap transit Entry, Guía de instalación

Figura 3.7 Programa para el cobro de peaje

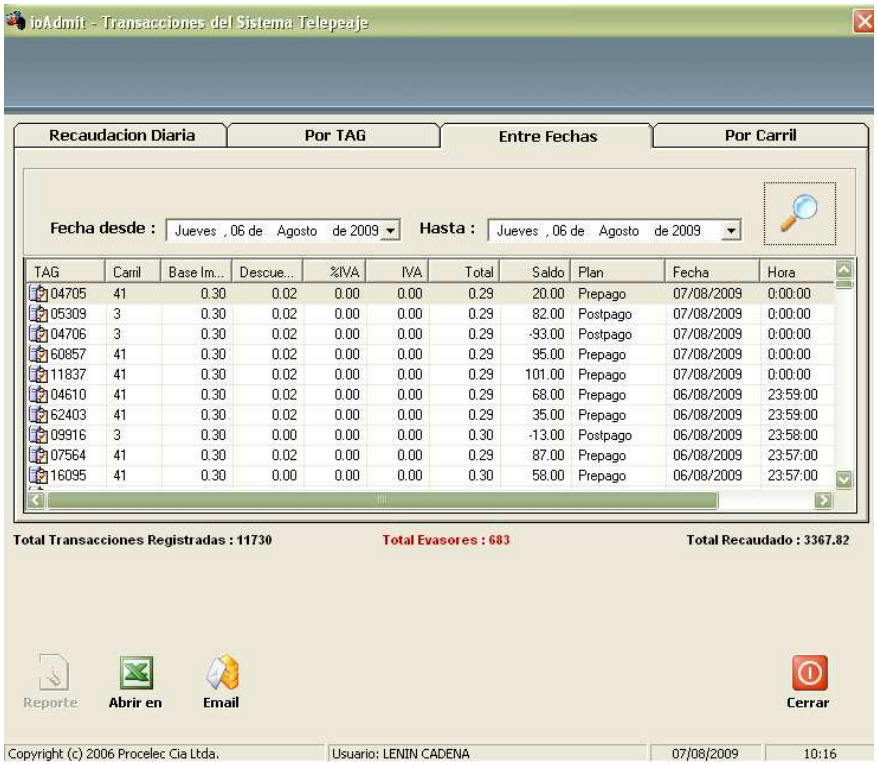


Figura 3.8 Interfaz de Usuario Peaje Túnel Guayasamín

Este sistema tiene 4 menús:

Recaudación Daria en donde se obtiene el total de autos que utilizan el telepeaje y disponen de un TAG.

Por TAG con esta opción obtenemos el número de pasadas que hizo un auto en el rango de fecha seleccionado.

Entre Fechas con este menú y en el rango de fecha que se seleccione se puede observar cuantas transacciones se realizaron, cuantos evasores hubo y el total recaudado. El la figura anterior por ejemplo se tiene que el 6/Agosto/2009 hubo 683 invasores[73].

Por Carril, en este menú se puede observar cuantos autos pasan por el carril en la fecha seleccionada.

[73] Usuarios de la autopista que utilizan al carril del telepeaje sin tener un TAG

CAPÍTULO 4

4. PRUEBAS, CONCLUSIONES Y RECOMENDACIONES

4.1 PRUEBAS

Para realizar las pruebas se colocó el DIMOVIL en un auto primero con voltaje de 5 V y se probó a distintas velocidades, el cual el cobro del peaje de una manera satisfactoria se obtuvo cuando el auto estaba a una distancia de 5 m. se realizó el mismo procedimiento alimentando los dispositivos con 9 V y dándonos como resultado el cobro de peaje cuando el auto se encontraba a 10 metros aproximadamente.

VOLTAJE (V)	VELOCIDAD (Km/h)	DISTANCIA (m)
5	20	5
	40	5
	60	5
	80	5
9	20	10
	40	10
	60	10
	80	10

Tabla 4.1 Potencia versus Distancia

4.2 CONSTRUCCIÓN Y COSTOS

En esta sección se realizará el estudio de costo de la construcción y montaje del prototipo diseñado. Los precios mostrados corresponden al mes de Mayo del 2009. Con estos datos se puede calcular un valor aproximado de cada dispositivo que va en el auto y del prototipo que se encuentra en la caseta.

En la tabla 4.2 y 4.3 se presentan la lista de los elementos necesarios para la construcción de los prototipos.

DIFIJO			
ELEMENTO	CANTIDAD	COSTO UNITARIO (Dólares Americanos)	COSTO TOTAL (Dólares Americanos)
CAJA NEGA	1	1.65	1,65
MICROCONTROLADOR PIC18F4550-SMD	1	11.25	11,25
CAPACITOR 1000uF	1	0.20	0,2
CAPACITOR 100uF	2	0.12	0,24
CAPACITOR 47uF	1	0.08	0,08
CAPACITOR 0.1uF - SMD	4	0.09	0,36
CAPACITOR 22pF - SMD	2	0.09	0,18
DIODO 1N4007 - SMD	1	0.1	0,1
DIODO 1N4148 - SMD	1	0.1	0,1
TRANSISTORES 2N3904 - SMD	2	0.09	0,18
RESISTENCIAS 10 KΩ - SMD	5	0.03	0,15

RESISTENCIAS 100 KΩ - SMD	2	0.03	0,06
RESISTENCIAS 220Ω - SMD	3	0.03	0,09
REGULADORES DE VOLTAJE CI7805 Y CI7809	2	0.45	0,9
CRYSTAL 20 MHz	1	0.55	0,55
LED VERDE, NARANJA Y ROJO	3	0.1	0,3
MODULOS DE RF TX Y RX	2	9	18
CONECTOR USB TIPO B	1	0.80	0,8
CONECTOR VCC	1	0.45	0,45
CABLE USB TIPO B	1	2	2
TOTAL			37,64 Dólares

Tabla 4.2 Costos DIFIJO

DIMOVIL			
ELEMENTO	CANTIDAD (Dólares Americanos)	COSTO UNITARIO (Dólares Americanos)	COSTO TOTAL (Dólares Americanos)
CAJA NEGA	1	1.65	1,65
MICROCONTROLADOR PIC18F2550-SMD	1	10.10	10,10
CAPACITOR 1000uF	1	0.20	0,20
CAPACITOR 100uF	2	0.12	0,24
CAPACITOR 0.1uF - SMD	4	0.09	0,36
CAPACITOR 22pF - SMD	2	0.09	0,18

DIODO 1N4007 - SMD	1	0.1	0,10
DIODO 1N4148 - SMD	1	0.1	0,10
TRANSISTORES 2N3904 - SMD	2	0.09	0,18
RESISTENCIAS 10 KΩ - SMD	7	0.03	0,21
RESISTENCIAS 100 KΩ - SMD	2	0.03	0,06
RESISTENCIAS 220Ω - SMD	3	0.03	0,09
REGULADORES DE VOLTAJE CI7805 Y CI7809	2	0.45	0,90
CRYSTAL 20 MHz	1	0.55	0,55
LED VERDE, NARANJA Y ROJO	3	0.1	0,30
MODULOS DE RF TX Y RX	2	9	18,00
MEMORIA 24LC08B	1	4	4
SOCALO	1	0.8	0,80
CONECTOR VCC	1	0.45	0,45
TOTAL			38,47 Dólares

Tabla 4.3 Costos DIMOVIL

CANTIDAD	DESCRIPCIÓN	P. UNITARIO (Dólares Americanos)	P.TOTAL (Dólares Americanos)
1	PANEL F1 DS+SM REF SME 4175	24,7	24,7
1	TODING PLOTER	10	10
1	SOLDADO PLACA CASETA_PCB	4,25	4,25
1	SOLDADO PLACA AUTO_PCB	3,5	3,5
			42,45 Dólares

Tabla 4.4 Costos PLACAS

Las placas fueron construidas por la empresa SME Elektronik (Cuenca), los detalles del costo se muestran en la tabla 4.4

4.3 CONCLUSIONES

- Se logró implementar tarjetas electrónicas capaces de transmitir, recibir y procesar datos en el cobro de peaje electrónico. Con estas tarjetas se logra obtener una solución económica y confiable para este tipo de prueba que en la actualidad varias empresas utilizan en los telepeajes.
- Para transferir datos entre el DIFIJO y el computador se utiliza el estándar USB logando de esta manera conocer este tipo de tecnología que hoy en día es muy útil y que en la industria puede ser en ciertos casos de gran ayuda ya lograr reducir costos.
- Debido a las características internas del microcontrolador escogido para el DIFIJO, no fue necesario agregar módulos adicionales, tales como para la transferencia de datos mediante USB, logrando ahorro en la implementación de la tarjeta y que el PCB sea de menores dimensiones.
- Se aplicaron las bases adquiridas y se mejoraron los conocimientos en Lenguaje C, que es un programador de alto nivel para desarrollar el Firmware de los microcontroladores
- El sistema diseñado permite minimizar el tiempo que un vehículo pasa por el peaje, se evita congestionamientos y además optimiza recursos de la empresa que administra el peaje.
- En la utilización de este sistema, es muy difícil controlar a los evasores del peaje. Porque un auto que no dispone de un TAG puede pasar por el carril designado para el telepeaje y así evadir el pago.

- Las comunicaciones inalámbricas móviles se pueden utilizar en cualquier campo de la electrónica. Estas comunicaciones se lo puede realizar mediante Radio Frecuencia, Bluetooth, infrarrojo. Siendo la mejor para estos casos la comunicación mediante RF ya que no requiere línea de vista y se enlaza a mayor distancia que en las otras tecnologías.
- En el pago de peaje también se puede utilizar código de barras, pero es limitado con relación a la comunicación por RF sobre todo por la capacidad de almacenamiento que tienen la tecnología de barras.
- Los dispositivos desarrollados para el cobro de peajes, se puede utilizar en otras aplicaciones, por ejemplo en parqueaderos, para autenticar a un bus cuando pasa por una parada o permitir a un vehículo el ingreso a una casa.

4.4 RECOMENDACIONES

- Antes de realizar un diseño que utilice comunicaciones inalámbricas, hay que definir que tecnología inalámbrica se va a utilizar: RF, Bluetooth o infrarrojo. Una vez identificado cual se va a utilizar hay que investigar en el mercado que dispositivos se disponen, para no tener problemas en el momento de conseguir otro igual si un caso se dañe el módulo.
- Los TAGs activos son recomendables cuando se aplican en estaciones de peaje, ya que estos disponen de una batería interna y puede alcanzar grandes distancias.
- Cuando se utilice estos sistemas de cobro hay que hacerlo con la cooperación de la policía, de esta manera disminuir las infracciones que se comete cuando un auto pasa por el carril designado exclusivamente para el telepeaje y no dispone de un TAG.

- En el diseño y construcción de algún sistema, de preferencia se recomienda realizar primeramente el diseño completo del hardware para luego en base a este desarrollar o utilizar el software óptimo para el hardware ya diseñado y armado.

BIBLIOGRAFÍA

- ✓ González Vásquez, José Adolfo / Introducción a los microcontroladores: hardware, software y aplicaciones
- ✓ BERNAL, Iván. Comunicaciones Inalámbricas, Escuela Politécnica Nacional, Quito Ecuador, 2005
- ✓ Valdés, Fernando. Microcontroladores: Fundamentos y aplicaciones con PIC, Marcombo, España, 2007
- ✓ Pérez, Enrique. Microcontroladores PIC, Marcombo, España 2007
- ✓ García, Eduardo. Compilador C CCS y simulador proteus para microcontroladores PIC, Alfaomega, México 2008
- ✓ CORDOVA A., Manual de Usuario del Compilador PCW de CCS, C Compiler for Microchip PICmicro.
- ✓ ANGULO J., ROMERO S., ANGULO I., Microcontroladores PIC: Diseño práctico de aplicaciones. Segunda Parte: PIC 16F87X, Mc. Graw Hill Madrid
- ✓ CEVALLOS F., Curso de Programación de Visual Basic 6.0, 2006
- ✓ TOMASI, Wayne. "Sistemas de Comunicaciones Electrónicas". Editorial Prentice - Hall. 2003

DIRECCIONES ELECTRÓNICAS

- ✓ Introducción a los Microcontroladores

  http://biblioteca.uct.cl/tesis/rodrigo-venegas/tesis.pdf

  Tesis - Desarrollo de la Plataforma para el manejo de Robot a distancia por medio de RF

http://www2.ing.puc.cl/~iee3912/files/pic.pdf
Microcontroladores

http://www.olimex.cl/tutorial/tutorial1.pdf
Introducción al microcontrolador

http://www.aiu.edu/applications/DocumentLibraryManager/upload/Despradel%20Novas%20Pe%C3%B1a.pdf
Microcontroladores: Arquitectura, programación y aplicación

http://server-die.alc.upv.es/asignaturas/LSED/2002-03/Micros/downloads/trabajo.pdf
Comparativa de microcontroladores actuales

- ✓ Microcontroladores PIC

http://www.ibars.com/assets/presentations/PIC18%20(sph).pdf
Comenzando con los PIC18: Arquitectura, Set de Instrucciones, Interrupciones, Periféricos y Características especiales

http://cde05.etse.urv.es/pub/pdf/524pub.pdf
Prácticas PIC basadas en máquina de vending

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1335&dDocName=en010280
Data sheet PICs

- ✓ Microcontroladores ATMEL

http://www.atmel.com/dyn/resources/prod_documents/doc4316.pdf
Atmel 8051 Microcontrollers Hardware Manual

http://www.atmel.com/dyn/products/datasheets.asp?family_id=604
Data Sheet ATMEL

- ✓ Comunicación serial y protocolo I^2C

http://www.ate.uniovi.es/fernando/Doc2003/SED/SCI_sincrono.pdf
COMUNICACIÓN SERIE PARA SISTEMAS BASADOS EN MICROCONTROLADORES PIC

http://www.i-micro.com/pdf/articulos/rs-232.pdf
INGENIERIA EN MICROCONTROLADORES

http://www.comunidadelectronicos.com/articulos/i2c.htm
El bus I2C

http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf
THE I^2C-BUS SPECIFICATION

http://www.nxp.com/documents/application_note/AN10216.pdf
AN10216-01 I2C MANUAL

- ✓ Memoria 24LC08B

http://ww1.microchip.com/downloads/en/devicedoc/21710c.pdf
Data Sheets 24AA08/24LC08B

- ✓ Telepeajes

http://es.wikipedia.org/wiki/Cobro_electr%C3%B3nico_de_peajes
http://es.wikipedia.org/wiki/Autopista_Costanera_Norte_(Santiago_de_Chile)

- ✓ Elementos Electrónicos

Transistores

http://mit.ocw.universia.net/6.301/NR/rdonlyres/Electrical-Engineering-and-Computer-Science/6-301Solid-State-CircuitsSpring2003/2A9D6D81-74A6-4FEA-BE44-AF521D5F9CA2/0/2N3904Fairchild.pdf
http://fisica.usach.cl/~weblab/Documentos/fisica/acdc/Transistores1.pdf

Fuentes de alimentación

http://www.hispavila.com/3ds/lecciones/lecc3.htm

Diodos

http://xuyang.manufacturer.globalsources.com/si/6008811439325/pdtl/Diode-diode/1025717902/M1-M7-SMD-1A.htm
http://www.suntan.com.hk/pdf/Rectifier-Diode/SMD.pdf

LEDs

http://www.superpbenavides.com/catalogo/componentes%20pasivos/optoelectronicos/Como-disenar-LED.pdf

✓ Módulos Inalámbricos

http://www.linxtechnologies.com/Documents/AN-00130.pdf
APPLICATION NOTE AN-00130 Modulation Techniques For Low-Cost RF Data Links

http://www.linxtechnologies.com/Documents/TXM-xxx-LC_Data_Guide.pdf
LC SERIES TRANSMITTER MODULE DATA GUIDE

http://www.linxtechnologies.com/Documents/RXM-xxx-LC-S_Data_Guide.pdf
LC SERIES RECEIVER MODULE DATA GUIDE

http://www.laipac.com/easy_434_esp.htm
Transmitter and Receiver LAIPAC

http://www.laipac.com/Downloads/Easy/tlp434a.pdf
TLP434A - RF ASK Hybrid Modules for Radio Control

http://www.laipac.com/Downloads/Easy/rlp434.pdf
RLP434A - RF ASK Hybrid Modules for Radio Control

http://www.laipac.com/Downloads/Easy/tlp_power.pdf
Gráfico de Potencia de los módulos LAIPAC

http://www.e-madeinchn.com/TransimittingModules.html
Wireless Radio Frequency Transmitting Modules

http://www.e-madeinchn.com/ReceiverModules.html
Receiver Modules

http://www.tuxen.us/nedapavi/CRS/documents/Entry_InstallGuide_E.pdf
NEDAP TRANSIT Entry: Guía de Instalación

www.securitychile.com/Folleto_TRANSIT_Entry_Spanish.pdf
Nedap, acceso manos libres

- ✓ Antenas

http://www.linxtechnologies.com/Documents/AN-00150.pdf
APPLICATION NOTE AN-00150. Use and Design of T-Attenuation Pads

http://www.linxtechnologies.com/Documents/AN-00500.pdf
APPLICATION NOTE AN-00500. Antennas: Design, Application, and Performance

http://www.solred.com.ar/lu6etj/tecnicos/loop/antena_de_cuadro.htm
La antena de cuadro o "Magnetic Loop"

- ✓ Visual Basic

http://mat21.etsii.upm.es/ayudainf/aprendainf/VisualBasic6/vbasic60.pdf
Aprenda Visual Basic 6.0

http://www.cs.us.es/cursos/ai-2003/VB/VB.pdf
Introducción al Desarrollo de Aplicaciones con Visual Basic

ANEXO A

FIRMWARE DIFIJO

```
//MODULO PRINCIPAL

//*****************************Directivas*******************************
*
#include <HardwareProfile.h> //Incluye el fichero del hardware: Puertos,
interrupciones, temporizadores
#include <usb_cdc.h>         //Driver que permite utilizar una clase de
dispositivo CDC USB,
                             //emulando un dispositivo RS232 y lo muestra
como un puerto com en Windows
#include <string.h>          //Librería para el manejo de cadenas de texto
#include <stdlib.h>          //Librería funciones internas del compilador
#include <ProtocoloRF.c>     //Incluye el fichero para el manejo de la
comunicación inalámbrica
#use fast_io(B)              //Gestión de puertos
#use fast_io(C)              //Gestión de puertos


//*************************VARIABLES GLOBALES*************************

//*************************Procesa comandos*****************************
unsigned int8 state_comandos=0;

//*************************Procesa trama USB****************************
         char temp_usb;
         char buf_rx_USB[len_buf]="";
unsigned int8 state_usb=0;
unsigned int8 x=0;
         int1 peaje_manual=0;

//***********************Cobro Peaje Automatico*************************
unsigned int8 state_cobrar_peaje_automatico=0;
         int1 timeout=0;
unsigned int8 contador_timer=0;
unsigned int8 contador_timeout=0;
unsigned int8 ultimo_comando_recibido=0;


//************************TAREAS QUE SE EJECUTAN EN MAIN****************
void procesa_ComandosRF_task(void);
void usb_debug_task(void);
```

```
void procesa_TramaUSB_task(void);
void cobrar_Peaje_Automatico_task(void);


//******Interrupción cuando recibe datos de la comunicación inalámbrica***
#INT_RDA
void serial_isr()
{
   buf_rx_RF[cont_rx] =getc();                    //Implementa memoria
circular para recepcion de caracteres en interrupcion
   cont_rx = (cont_rx+1) % len_buf_rx;
}


//*********************Interrupción de desbordamiento del timer 0*********
#INT_RTCC
void Temporizador_timer0()
{
   ++contador_timeout;
   ++contador_timer;
   if(contador_timeout>=3){
      timeout=1;
      contador_timeout=0;}
   set_timer0(18660);              //para 48Mhz PS256 1s
}


//***************Funcion para escribir en la memoria no volatil***********
void Escribir_EEPROM(int n, float data)
{
   int i;
   for (i = 0; i < 4; i++)
   write_eeprom(i + n, *(((int8*)&data + i) )) ;
}


//******************Funcion para leer en la memoria no volatil***********
float Leer_EEPROM(int n)
{
   int i;
   float data;
   for (i = 0; i < 4; i++)
     *(((int8*)&data + i)) = read_eeprom(i + n);
   return(data);
}


//*********************Entrada principal del programa*****************
void main()
{
   InitializeSystem(); //Inicialización del hardware y las variables

   usb_init_cs();
```

```
   tarifa_liviano=Leer_EEPROM(mem_liviano);
   tarifa_pesado=Leer_EEPROM(mem_pesado);
   tarifa_muy_pesado=Leer_EEPROM(mem_muy_pesado);
   tarifa_especial=Leer_EEPROM(mem_especial);
   EstadoEquipoPeaje=read_eeprom(dirEstadoEquipoPeaje);


   if(EstadoEquipoPeaje==peajeAutomatico)
   {
      enable_interrupts(INT_RTCC);
   }
   else
   {
      disable_interrupts(INT_RTCC);
   }


   while (TRUE) //Bucle principal
   {
      usb_task();
      procesa_TramaUSB_task();
      procesa_TramaRF_Recibida();         //Se encarga de recibir y
procesar el dato que llega por RF
      procesa_ComandosRF_task();
      cobrar_Peaje_Automatico_task();
   }

}


//****************Stack y funciones USB:*********************************
//***Se ejecutan cuando se conecta el dispositivo a un puerto USB del PC**
void usb_debug_task(void)
{
   static int8 last_connected;
   static int8 last_enumerated;
   int8 new_connected;
   int8 new_enumerated;

   new_connected=usb_attached();
   new_enumerated=usb_enumerated();

   if (new_connected && !last_connected)
      bit_set(m_usb,0);
   if (!new_connected && last_connected)
      bit_clear(m_usb,0);
   if (new_enumerated && !last_enumerated)
      bit_set(m_usb,1);
   if (!new_enumerated && last_enumerated)
      bit_clear(m_usb,1);

   last_connected=new_connected;
   last_enumerated=new_enumerated;
}
```

```
//*******************************Procesa trama USB*********************
void procesa_TramaUSB_task(void)
{
   static unsigned int8 tarifas=0;
   usb_debug_task();
   if (usb_cdc_kbhit())
   {
      temp_usb=usb_cdc_getc();
      switch(state_usb)
      {
         case free:           switch(temp_usb)
                              {
                                 case sol_ID:
enviar_trama_RF(sol_ID);
                                                                  break;

                                 case  sol_Saldo:
enviar_trama_RF(sol_saldo);
                                                                  break;

                                 case  Sol_TarifasPeaje:
enviar_trama_USB(Resp_TarifasPeaje);
                                                                  break;

                                 case cobrar_Peaje:
enviar_trama_RF(cobrar_peaje);

if(estadoEquipoPeaje==peajeManual)
                                                                  {
                                                                     peaje_manual=1;

output_bit(LEDpase,0);

output_bit(LEDpare,1);
                                                                  }
                                                                  break;

                                 case Sol_IDcaseta:
enviar_trama_USB(Resp_IDcaseta);
                                                                  break;

                                 case Sol_equipoPeaje:
enviar_trama_USB(Resp_equipoPeaje);
                                                                  break;

                                 default:
state_usb=temp_usb;
                              }
                              break;

         case Recargar_Saldo: buf_rx_USB[x]=temp_USB;
                              ++x;
                              buf_rx_USB[x]=null;
                              if(Temp_USB==sep_trama)
```

```
                                {
                                   buf_rx_USB[x-1]=null;
                                   strcopy(Datos_Trama_USB,buf_rx_USB);
                                   enviar_trama_RF(Recargar_Saldo);
                                   x=0;state_USB=free;
                                }
                                else
                                {
                                   if(x>=len_buf){x=0;state_USB=free;}    //
trama con error
                                }
                                break;

         case Nuevas_Tarifas: buf_rx_USB[x]=temp_USB;
                              ++x;
                              buf_rx_USB[x]=null;
                              if(Temp_USB==sep_trama)
                              {
                                 buf_rx_USB[x-1]=null;
                                 switch(tarifas)
                                 {
                                    case 0:
tarifa_liviano=atof(buf_rx_USB);

Escribir_EEPROM(mem_liviano,tarifa_liviano);
                                                tarifas=1;
                                                break;

                                    case 1: tarifa_pesado=atof(buf_rx_USB);

Escribir_EEPROM(mem_pesado,tarifa_pesado);
                                                tarifas=2;
                                                break;


                                    case 2:
tarifa_muy_pesado=atof(buf_rx_USB);

Escribir_EEPROM(mem_muy_pesado,tarifa_muy_pesado);
                                                tarifas=3;
                                                break;


                                    case 3:
tarifa_especial=atof(buf_rx_USB);

Escribir_EEPROM(mem_especial,tarifa_especial);
                                                tarifas=0;
                                                enviar_trama_USB(ACK);
                                                state_USB=free;
                                                break;

                                    default: tarifas=0,state_USB=free;
                                 }
                                   x=0;
                                }
                                else
```

```
                                    {
if(x>=len_buf){x=0;state_USB=free;tarifas=0;}    // trama con error
                                    }
                                break;

         case nuevoEstadoPeaje: buf_rx_USB[x]=temp_USB;
                                ++x;
                                buf_rx_USB[x]=null;
                                if(Temp_USB==sep_trama)
                                {
                                    buf_rx_USB[x-1]=null;
                                    if (buf_rx_USB[x-2]=='2')
                                    {
                                       estadoEquipoPeaje=peajeManual;
                                       disable_interrupts(INT_RTCC);
                                    }
                                    if (buf_rx_USB[x-2]=='3')
                                    {
                                       estadoEquipoPeaje=peajeAutomatico;
                                       state_cobrar_peaje_automatico=0;
                                       timeout=0;
                                       contador_timeout=0;
                                       contador_timer=0;
                                    }

write_eeprom(dirEstadoEquipoPeaje,estadoEquipoPeaje);
                                    enviar_trama_USB(Resp_equipoPeaje);
                                    x=0;state_USB=free;
                                    output_bit(LEDpare,1);
                                    output_bit(LEDpase,0);
                                }
                                else
                                {
                                    if(x>=len_buf){x=0;state_USB=free;}
// trama con error
                                }
                                break;

         default:               printf("Comando_USB_desconocido\r\n");
                                state_usb=free;
                                temp_usb=free;
                                break;
      }
   }

}


//*********************Procesa los datos de la comunicación RF************
void procesa_ComandosRF_task(void)
{
   switch(state_comandos)
   {
```

```
      case  free:
if(OK_trama){OK_trama=free;state_comandos=tipo_comando;}
                        break;

      case  resp_ID:    tipo_vehiculo=atoi(datos_trama_RF);
                        enviar_trama_USB(resp_ID);
                        state_comandos=free;
                        ultimo_comando_recibido=resp_ID;
                        break;

      case  resp_saldo: enviar_trama_USB(resp_Saldo);
                        state_comandos=free;
                        ultimo_comando_recibido=resp_saldo;
                        if(peaje_manual)
                        {
                           output_bit(LEDpase,1);
                           output_bit(LEDpare,0);
                           peaje_manual=0;
                        }
                        break;

      case ACK:         enviar_trama_USB(ACK);
                        state_comandos=free;
                        break;

      default:          state_comandos=free;OK_trama=free;
   }
}


//*********************************Cobro del peaje************************
void cobrar_Peaje_Automatico_task(void)
{
   if(estadoEquipoPeaje==peajeAutomatico)
   {
      switch(state_cobrar_peaje_automatico)
      {
         case 0: enviar_trama_RF(Sol_ID);
                 timeout=0;
                 ultimo_comando_recibido=0;
                 output_bit(LEDpare,1);
                 output_bit(LEDpase,0);
                 state_cobrar_peaje_automatico=1;
                 contador_timeout=0;
                 enable_interrupts(INT_RTCC);
                 break;

         case 1: if(timeout)
                 {
                  state_cobrar_peaje_automatico=0;
                  timeout=0;
                  break;
                 }

if((ultimo_comando_recibido==resp_ID)&&(contador_timeout>=1))
```

```
                {
                 enviar_trama_RF(cobrar_peaje);
                 state_cobrar_peaje_automatico=2;
                 ultimo_comando_recibido=0;
                 contador_timeout=0;
                }
                break;

        case 2: if(timeout)
                {
                 state_cobrar_peaje_automatico=0;
                 timeout=0;
                 break;
                }
                if(ultimo_comando_recibido==Resp_Saldo)
                {
                 output_bit(LEDpase,1);
                 output_bit(LEDpare,0);
                 state_cobrar_peaje_automatico=3;
                 ultimo_comando_recibido=0;
                 contador_timer=0;
                 contador_timeout=0;
                }
                break;

        case 3: if(contador_timer>=7)
                {
                 state_cobrar_peaje_automatico=0;
                 timeout=0;
                 contador_timeout=0;
                }
                break;

        default: state_cobrar_peaje_automatico=0;
                 timeout=0;
                 contador_timeout=0;
                 contador_timer=0;
                 break;
     }
   }
}



//*****************PROGRAMA PARA LA RADIOFRECUENCIA**********************
#define  len_buf_rx     40       //tamaño del bufer principal
#define  len_buf        12       //tamaño de todos los bufers de
procesamiento


//************************Tipo de vehículos*******************************
#define  liviano           1
#define  pesado            2
#define  muy_pesado        3
```

```
#define  especial          4

#define  mem_liviano       1
#define  mem_pesado        5
#define  mem_muy_pesado    9
#define  mem_especial      13

#define  peajeManual       2
#define  peajeAutomatico   3
#define  dirEstadoEquipoPeaje 0x50
#rom     int8 0xf00050={peajeManual}


//*********************Tipo de tramas************************************
#define  free              0
#define  Sol_TarifasPeaje  1
#define  Resp_TarifasPeaje 2
#define  Sol_ID            3
#define  Resp_ID           4
#define  Sol_Saldo         5
#define  Resp_Saldo        6
#define  Recargar_Saldo    7
#define  Cobrar_Peaje      8
#define  Nuevas_Tarifas    9
#define  Sol_IDcaseta      10
#define  Resp_IDcaseta     11
#define  Sol_equipoPeaje   12
#define  Resp_equipoPeaje  13
#define  nuevoEstadoPeaje  14
#define  ACK               15


//*********************Procesamiento de la tama RF***********************
#define  inicio_frame      0
#define  IDdestino_frame   1
#define  IDorigen_frame    2
#define  TipoComando_frame 3
#define  data_frame        4


//******************Variables globales para uso en Main******************
         char  ID_origen_RF[len_buf]="";         //ID del equipo envia la
trama
unsigned int8  tipo_comando=free;
         char  Datos_trama_RF[len_buf_rx]="";    //Bufer para la recepcion
de los datos de la trama
         char  Datos_trama_out_RF[len_buf]="";   //Datos para el envio por
RF
         char  Datos_Trama_USB[len_buf]="";
         int1  OK_trama=0;                       //Una trama se recibio con
exito y debe ser procesada
unsigned int8  m_usb=free;
unsigned int8  tipo_vehiculo;                    //Tipo de vehiculo con el
que se esta comunicando
```

```
         float32  tarifa_liviano;                //Valores con decimales
que tengan punto
         float32  tarifa_pesado;
         float32  tarifa_muy_pesado;
         float32  tarifa_especial;
unsigned int8  estadoEquipoPeaje=0;             //2 = Peaje Manual; 3 =
Peaje Automatico



//********************Para el procesamineto de la trama RF***************
         char  buf_rx_RF[len_buf_rx]="";         //Bufer circular para
almacenar los datos de RF
unsigned int8  cont_rx=0;                       //Contador de datos
recibidos
unsigned int8  read_rx=0;                       //Contador de datos
procesados
unsigned int8  state_protocolo=0;               //Variable que barre toda
la trama separando la informacion
         char  Temp=0;                          //Variable temporal de
procesamiento

      char  ID_caseta [9]= {'C','A','S','E','T','A',' ','1'};
//Identificacion unica de la caseta
const char  ini_trama='#';
//identificador inicio trama
const char  out_trama='*';                                      //Byte
inicio
const char  fin_trama='\r';
//identificador fin trama
const char  sep_trama='/';
//identificador separador datos de trama
      char  Broadcast[6]={'1','1','1','1','1'};                 //Trama
broadcast


//***********************Función para enviar una trama hacia RF************
void enviar_trama_RF(unsigned int8 type_command)
{
   output_bit(LEDstatus,1);
   putchar(0xAA);    //Bits de sincronización
   putchar(0xAA);    //Bits de sincronización
   putchar('P');     //Bits de sincronización
   putchar('r');     //Bits de sincronización
   putchar(' ');     //Bits de sincronización
   switch(type_command)
   {
      case sol_ID:
printf("%c%s/%s/%c/%c\n",out_trama,Broadcast,ID_caseta,type_command,fin_tra
ma);
                           break;

      case sol_Saldo:
printf("%c%s/%s/%c/%c\n",out_trama,ID_origen_RF,ID_caseta,type_command,fin_
trama);
```

```
                              break;

      case Recargar_Saldo:
printf("%c%s/%s/%c/%s/%c\n",out_trama,ID_origen_RF,ID_caseta,type_command,D
atos_Trama_USB,fin_trama);
                              break;

      case cobrar_Peaje:    switch(tipo_vehiculo)
                            {
                               case liviano:
sprintf(Datos_trama_out_RF,"%f",tarifa_liviano);
                                                    break;

                               case pesado:
sprintf(Datos_trama_out_RF,"%f",tarifa_pesado);
                                                    break;

                               case muy_pesado:
sprintf(Datos_trama_out_RF,"%f",tarifa_muy_pesado);
                                                    break;

                               case especial:
sprintf(Datos_trama_out_RF,"%f",tarifa_especial);
                                                     break;

                               default:
tipo_vehiculo=0,Datos_trama_out_RF[0]=null;
                            }

printf("%c%s/%s/%c/%s%c\n",out_trama,ID_origen_RF,ID_caseta,type_command,Da
tos_trama_out_RF,fin_trama);
                              break;

      default: Printf("enviar_trama_RF Inexistente");
   }
   output_bit(LEDstatus,0);

}


//**********************Procesa la trama recibida por RF*****************
void procesa_TramaRF_Recibida(void)
{
static unsigned int8 a;                       //contador temporal
static char  rx_temporal[len_buf]="";         //bufer de comando temporal
   if(read_rx!=cont_rx)                       //pregunta si hay datos que
recibir
   {
      do
      {
         Temp=buf_rx_RF[read_rx];
         read_rx = (read_rx+1) % len_buf_rx;
         switch(state_protocolo)
         {
```

```
            case inicio_frame:
if(Temp==ini_trama){state_protocolo=IDdestino_frame;}   //Busca inicio de
trama
                                    break;

            case IDdestino_frame:   rx_temporal[a]=Temp;
                                    ++a;
                                    rx_temporal[a]=null;
                                    if(Temp==sep_trama)
                                    {
                                       rx_temporal[a-1]=null;
                                       a=0;
                                       if(!strcmp(rx_temporal,ID_caseta))
//Revisa si trama es para este vehiculo
                                       {
                                          state_protocolo=IDorigen_frame;
                                       }
                                       else
                                       {
                                          state_protocolo=inicio_frame;
//trama NO es para este vehiculo
                                       }
                                    }
                                    else
                                    {

if(a>=len_buf){a=0;state_protocolo=inicio_frame;}    //Trama con error
                                    }
                                    break;

            case IDorigen_frame:    rx_temporal[a]=Temp;
                                    ++a;
                                    rx_temporal[a]=null;
                                    if(Temp==sep_trama)
                                    {
                                       rx_temporal[a-1]=null;
                                       strcopy(ID_origen_RF,rx_temporal);

a=0;state_protocolo=TipoComando_frame;
                                    }
                                    else
                                    {

if(a>=len_buf){a=0;state_protocolo=inicio_frame;}    //Trama con error
                                    }
                                    break;

            case TipoComando_frame: rx_temporal[a]=Temp;
                                    ++a;
                                    rx_temporal[a]=null;

if((Temp==sep_trama)||(Temp==fin_trama))
                                    {
                                       rx_temporal[a-1]=null;
                                       a=0;
```

```
                                    tipo_comando=rx_temporal[0];
                                    if(Temp==fin_trama)
                                    {
                                       state_protocolo=inicio_frame;
                                       OK_trama=1;
                                    }
                                    else
                                    {
                                       state_protocolo=data_frame;
                                    }
                                 }
                                 else
                                 {
if(a>=len_buf){a=0;state_protocolo=inicio_frame;}    //Trama con error
                                 }
                                 break;

            case  data_frame:    datos_trama_RF[a]=Temp;
                                 ++a;
                                 datos_trama_RF[a]=null;
                                 if(Temp==fin_trama)
                                 {
                                    datos_trama_RF[a-1]=null;
                                    a=0;
                                    OK_trama=1;
                                    state_protocolo=inicio_frame;
//Trama se recibio con exito
                                 }
                                 else
                                 {
if(a>=20){a=0;state_protocolo=inicio_frame;}    //Trama con error
                                 }
                                 break;

            default:             state_protocolo=inicio_frame;a=0;
                                 OK_trama=0;
         }
      }
      while(read_rx!=cont_rx);              //No sale de bucle si no
procesa toda la data
   }
}


//**********************Envía trama USB*********************************
void enviar_trama_USB(unsigned int8 type_command_USB)
{
   switch(type_command_USB)
   {
      case resp_ID:
if((m_usb==3)&&(input(USBsense))){printf(usb_cdc_putc,"%c/%s/%u/\r",type_co
mmand_USB,ID_origen_RF,tipo_vehiculo);}
                                 break;
```

```
      case resp_Saldo:
if((m_usb==3)&&(input(USBsense))){printf(usb_cdc_putc,"%c/%s/%s\r",type_com
mand_USB,ID_origen_RF,datos_trama_RF);}
                                 break;

      case  Resp_TarifasPeaje:
if((m_usb==3)&&(input(USBsense))){printf(usb_cdc_putc,"%c/%f/%f/%f/%f"type_
command_USB,tarifa_liviano,tarifa_pesado,tarifa_muy_pesado,tarifa_especial)
;}
                                 break;

      case Resp_IDcaseta:
if((m_usb==3)&&(input(USBsense))){printf(usb_cdc_putc,"%c/%s/\r",type_comma
nd_USB,ID_caseta);}
                                 break;

      case Resp_equipoPeaje:
if((m_usb==3)&&(input(USBsense))){printf(usb_cdc_putc,"%c/%u/\r",type_comma
nd_USB,estadoEquipoPeaje);}
                                 break;

      case ACK:
if((m_usb==3)&&(input(USBsense))){printf(usb_cdc_putc,"%c",type_command_USB
);}
                                 break;

      default:
if((m_usb==3)&&(input(USBsense))){printf(usb_cdc_putc,"Error Trama
USB\r");}
   }
}
```

```
#include <18F4550.h>
#device adc=8

#fuses   HSPLL                   //High Speed Crystal/Resonator with PLL
enabled
#fuses   PLL5                    //Divide By 5(20MHz oscillator input)


#FUSES NOWDT                    //No Watch Dog Timer
#FUSES NOPROTECT                //Code not protected from reading
#FUSES NOBROWNOUT                 //Reset when brownout detected
#FUSES NOPUT                    //No Power Up Timer
#FUSES NOCPD                    //No EE protection
#FUSES STVREN                   //Stack full/underflow will cause reset
#FUSES NODEBUG                  //No Debug mode for ICD
```

```
#FUSES NOLVP                     //Low Voltage Programming on B3(PIC16) or
B5(PIC18)
#FUSES NOWRT                    //Program memory not write protected
#FUSES NOWRTD                   //Data EEPROM not write protected
#FUSES NOIESO                     //Internal External Switch Over mode
enabled
#FUSES FCMEN                    //Fail-safe clock monitor enabled
#FUSES NOPBADEN                   //PORTB pins are configured as analog
input channels on RESET
#FUSES NOWRTC                   //configuration not registers write
protected
#FUSES NOWRTB                   //Boot block not write protected
#FUSES NOEBTR                   //Memory not protected from table reads
#FUSES NOEBTRB                  //Boot block not protected from table reads
#FUSES NOCPB                    //No Boot Block code protection
#FUSES NOMCLR                     //Master Clear pin enabled
#FUSES NOLPT1OSC                  //Timer1 configured for low-power
operation
#FUSES NOXINST                  //Extended set extension and Indexed
Addressing mode disabled (Legacy mode)
#FUSES CPUDIV1                  //System Clock by 1
#FUSES USBDIV                   //USB clock source comes from PLL divide by
2
#FUSES VREGEN                   //USB voltage regulator enabled

#use delay(clock=48M)
#use rs232(baud=2400,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=8)


//*********************************************************************
*******************
// CONFIGURACION DE PUERTOS A USAR
//*********************************************************************
*******************


#define  LEDpase        PIN_B5
#define  LEDstatus      PIN_B4
#define  LEDpare        PIN_B3
#define  USBsense       PIN_B2

#define  ON    1
#define  OFF   0


Void InitializeSystem(void)
{
   setup_adc_ports(NO_ANALOGS|VSS_VDD);
   setup_adc(ADC_OFF);
   setup_spi(SPI_SS_DISABLED);
   setup_wdt(WDT_OFF);
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_comparator(NC_NC_NC_NC);
```

```
   setup_vref(FALSE);


   enable_interrupts(int_rda);                    //habilita int. de
recepcion serial

   enable_interrupts(global);                     //habilita
interrupciones globales

   set_tris_b(0b00000111);
   set_tris_c(0b10111111);

   output_bit(LEDpare,1);
   output_bit(LEDpase,0);
   output_bit(LEDstatus,0);


                                    }
```

ANEXO B

FIRMWARE DIMOVIL

```
//*************************CODIGO DIMOVIL*******************************

//************************Encabezados y Librerías************************
#include <PerfilesHardware.h>     //Configuracion del hardware
#include <2408.c>
#include <string.h>              //libreria menejar cadena de caracteres
#include <stdlib.h>
#include <ProtocoloRF.c>


//****************Manejo de puertos**************
//#use fast_io(a)
#use fast_io(b)
#use fast_io(c)


//************************VARIABLES GLOBALES*****************************

//************************Procesa los comandos***************************
const    int8  dir_mem_Saldo=2; //Dirección de la memoria donde se guarda
el saldo
unsigned int8  state_comandos=0;
unsigned int8  contador_timer=0;
         int1  b=0;


//********************Tareas (funciones) que se ejecutan en main**********
void procesa_comandos_task(void);
void estadoLED(void);
float Leer_EEPROM(int n);
void Escribir_EEPROM(int n, float data);


//************************Programa
principal*********************************
void main()
{
   InitializeSystem();
   init_ext_eeprom();
   Saldo_actual=Leer_EEPROM(dir_mem_Saldo);
   estadoLED();
   while (TRUE)
   {
      procesa_TramaRF_task();
```

```
      procesa_comandos_task();
   }

}


//********************Habilita las interrupciones del puerto serie*******
#INT_RDA
void serial_isr()
{
   buf_rx_RF[cont_rx] =getc();                  //Implementa memoria
circular para recepcion
   cont_rx = (cont_rx+1) % len_buf_rx;          //de caracteres en
interrupcion
}


//***************Habilita interrupciones del desbordamiento del Timer 0****
#INT_RTCC
void contador_tiempo()
{
   ++contador_timer;
   if(contador_timer>=50)
   {
      b=!b;
      output_bit(LEDsaldo_Low,b);
      contador_timer=0;
   }
   set_RTCC(61);            //20Mhz PS256 61=10ms
}


//********************Función para definir el tipo de comandos************
void procesa_comandos_task(void)
{
   switch(state_comandos)
   {
      case  free:
if(OK_trama){OK_trama=free;state_comandos=tipo_comando_RX;}
                           break;

      case  sol_ID:        enviar_trama_RF(resp_ID);
                           state_comandos=0;
                           break;

      case  sol_saldo:     enviar_trama_RF(resp_saldo);
                           state_comandos=0;
                           break;

      case  Recargar_Saldo: Saldo_temp=atof(datos_trama_RF);
                           Saldo_actual=Saldo_actual+saldo_temp;
                           Escribir_EEPROM(dir_mem_Saldo,Saldo_actual);
                           enviar_trama_RF(Resp_Saldo);
                           estadoLED();
                           state_comandos=0;
```

```
                              break;

      case  Cobrar_Peaje:     saldo_temp=atof(datos_trama_RF);
                              Saldo_actual=Saldo_actual-saldo_temp;
                              Escribir_EEPROM(dir_mem_Saldo,Saldo_actual);
                              enviar_trama_RF(Resp_Saldo);
                              estadoLED();
                              state_comandos=0;
                              break;

      default:          OK_trama=free;state_comandos=0;
   }
}


//**************************Función para controlar la señalización*******
void estadoLED(void)
{
   if(Saldo_actual>=2)
   {
      output_bit(LEDsaldo_OK,1);
      output_bit(LEDsaldo_Low,0);
      disable_interrupts(INT_RTCC);
   }
   else
   {
      enable_interrupts(INT_RTCC);
      output_bit(LEDsaldo_OK,0);
      output_bit(LEDsaldo_Low,1);
   }
}



//**************Función para escribir en la memoria EEPROM***************
void Escribir_EEPROM(int n, float data)
{
   int i;
   for (i = 0; i < 4; i++)
   write_ext_eeprom(i + n, *(((int8*)&data + i) )) ;
}


//**************Función para leer la memoria EEPROM*********************
float Leer_EEPROM(int n)
{
   int i;
   float data;
   for (i = 0; i < 4; i++)
     *(((int8*)&data + i)) = read_ext_eeprom(i + n);
   return(data);
}

//*****************************Programa para RF************************
```

```
#define  RF_ON    0
#define  RF_OFF   1
#define  len_buf_rx     40    //Tamaño del bufer para alamcenamiento de
datos RF
#define  len_buf        12    //Tamaño de todos los ID de procesamiento


//***************************Tipo de vehículos*************************
#define  liviano           1
#define  pesado            2
#define  muy_pesado        3
#define  especial          4


//***************************Tipo de tramas*******************************
#define  free              0
#define  Sol_TarifasPeaje  1
#define  Resp_TarifasPeaje 2
#define  Sol_ID            3
#define  Resp_ID           4
#define  Sol_Saldo         5
#define  Resp_Saldo        6
#define  Recargar_Saldo    7
#define  Cobrar_Peaje      8
#define  Nuevas_Tarifas    9
#define  Sol_IDcaseta      10
#define  Resp_IDcaseta     11
#define  ACK               12


//************************Para procesar tramas RF************************
#define  inicio_frame      0
#define  IDdestino_frame   1
#define  IDorigen_frame    2
#define  TipoComando_frame 3
#define  data_frame        4


//*****************Datos que se almacena para cada DIMOVIL***************
      char  ID_vehiculo [8]= {'P','A','T','-','4','1','5'};    //Placa del
vehículo
const int8  tipo_vehiculo = muy_pesado;                        //Tipo de
vehículo


//************************Constantes de uso general***********************
const char  ini_trama='*';                          //Se identifica cuando
inicia una trama
const char  out_trama='#';
const char  fin_trama='\r';                         //identifica cuando
termino la trama
const char  sep_trama='/';                          //separador de trama
      char  Broadcast[6]={'1','1','1','1','1'};    //Trama broadcast
```

```
//**********************Valores con decimales*****************************
   float32 Saldo_actual=0;
   float32 saldo_temp=0;


//**********************Variables globales para uso en Main**************
         char  ID_origen_RF[len_buf];           //ID del equipo que envía
la trama al vehículo
unsigned int8  tipo_comando_RX=0;               //0=free  1=solicitud ID
2=solicitud saldo 3=respuesta saldo 4= Recargar_Saldo
         char  datos_trama_RF[len_buf_rx]="";   //Bufer para la recepcion
de los datos de la trama
         int1  OK_trama=0;                      //Bandera indica que una
trama se recibio con exito y debe ser procesada


//**********************Variables para procesar la trama
RF******************
         char  buf_rx_RF[len_buf_rx]="";  //Bufer circular para
almaceamiento de datos que llegan por RF
unsigned int8  cont_rx=0;                  //Contador de datos recibidos
unsigned int8  read_rx=0;                  //Contador de datos procesados
unsigned int8  state_protocolo=0;          //Variable que barre toda la
trama separando la informacion
         char  Temp=0;                     //Variable temporal de
procesamiento


//************************Función para enviar una trama****************
void enviar_trama_RF(unsigned int8 type_command)
{
   disable_interrupts(int_rda);
   output_bit(LEDstatus,1);
   delay_ms(20);        //200
   putchar(0xAA);       //Caracteres de sincronización
   putchar(0xAA);
   putchar('P');
   putchar('r');
   putchar(' ');
   putchar('P');
   putchar('r');
   putchar(' ');


   switch(type_command)
   {
      case  resp_ID:
printf("%c%s/%s/%c/%u%c\n",out_trama,ID_origen_RF,ID_vehiculo,type_command,
tipo_vehiculo,fin_trama);
                        break;

      case  resp_saldo:
printf("%c%s/%s/%c/%3.2f%c\n",out_trama,ID_origen_RF,ID_vehiculo,type_comma
nd,Saldo_actual,fin_trama);
                        break;
```

```
      case ACK:
printf("%c%s/%s/%c/%c\n",out_trama,ID_origen_RF,ID_vehiculo,type_command,fi
n_trama);
                        break;

      default:          Printf("error_trama_RF default");
   }
   output_bit(LEDstatus,0);
   enable_interrupts(int_rda);
}


//************************Función que procesa una trama*****************
void procesa_TramaRF_task(void)
{
   static unsigned int8 a;                      //Contador temporal
   static char  rx_temporal[len_buf]="";        //Bufer de comando temporal
   if(read_rx!=cont_rx)                         //Chequea datos para
recibir
   {
      do
      {
         Temp=buf_rx_RF[read_rx];
         read_rx = (read_rx+1) % len_buf_rx;
         switch(state_protocolo)
         {
            case inicio_frame:
if(Temp==ini_trama){state_protocolo=IDdestino_frame;}   //Busca inicio de
trama
                                    break;

            case IDdestino_frame:   rx_temporal[a]=Temp;
                                    ++a;
                                    rx_temporal[a]=null;
                                    if(Temp==sep_trama)
                                    {
                                       rx_temporal[a-1]=null;
                                       a=0;
                                       if(!strcmp(rx_temporal,ID_vehiculo))
//Revisa si la trama es para este vehículo
                                       {
                                          state_protocolo=IDorigen_frame;
                                       }
                                       else
                                       {

if(!strcmp(rx_temporal,Broadcast))     //Revisa si la trama es un broadcast
                                          {
                                             state_protocolo=
IDorigen_frame;

                                          }
                                          else
                                          {
```

```
                                                state_protocolo=inicio_frame;
//Trama NO es para este vehículo
                                             }
                                          }
                                       }
                                       else
                                       {

if(a>=len_buf){a=0;state_protocolo=inicio_frame;}    //Trama con error
                                       }
                                       break;

            case IDorigen_frame:       rx_temporal[a]=Temp;
                                       ++a;
                                       rx_temporal[a]=null;
                                       if(Temp==sep_trama)
                                       {
                                          rx_temporal[a-1]=null;
                                          strcopy(ID_origen_RF,rx_temporal);

a=0;state_protocolo=TipoComando_frame;
                                       }
                                       else
                                       {

if(a>=len_buf){a=0;state_protocolo=inicio_frame;}    //Trama con error
                                       }
                                       break;

            case TipoComando_frame:    rx_temporal[a]=Temp;
                                       ++a;
                                       rx_temporal[a]=null;

if((Temp==sep_trama)||(Temp==fin_trama))
                                       {
                                          rx_temporal[a-1]=null;
                                          a=0;
                                          tipo_comando_RX=rx_temporal[0];
                                          if(Temp==fin_trama)
                                          {
                                             state_protocolo=inicio_frame;
                                             OK_trama=1;
                                          }
                                          else
                                          {
                                             state_protocolo=data_frame;
                                          }
                                       }
                                       else
                                       {

if(a>=len_buf){a=0;state_protocolo=inicio_frame;}    //Trama con error
                                       }
                                       break;
```

```
            case  data_frame:       datos_trama_RF[a]=Temp;
                                    ++a;
                                    datos_trama_RF[a]=null;
                                    if(Temp==fin_trama)
                                    {
                                       datos_trama_RF[a-1]=null;
                                       a=0;
                                       OK_trama=1;
                                       state_protocolo=inicio_frame;
//trama se recibio con exito
                                    }
                                    else
                                    {

if(a>=20){a=0;state_protocolo=inicio_frame;}    //Trama con error
                                    }
                                    break;

            default:                state_protocolo=inicio_frame;a=0;
                                    OK_trama=0;
         }
      }
      while(read_rx!=cont_rx);      //no sale de bucle si no procesa toda
la data
   }
}
```

```
#include <16F876a.h>
#fuses HS,NOWDT,PROTECT,PUT    //Ordenes para el programador
#use delay(clock=20M)
#use rs232(baud=2400,xmit=PIN_C6,rcv=PIN_C7)


//*********************************************************************
*******************
// CONFIGURACION DE PUERTOS A USAR
//*********************************************************************
*******************

#define  LEDsaldo_OK    PIN_B3
#define  LEDstatus      PIN_B2
#define  LEDsaldo_Low   PIN_B1
#define  boton1         PIN_B0


#define  ON    1
#define  OFF   0


Void InitializeSystem(void)
```

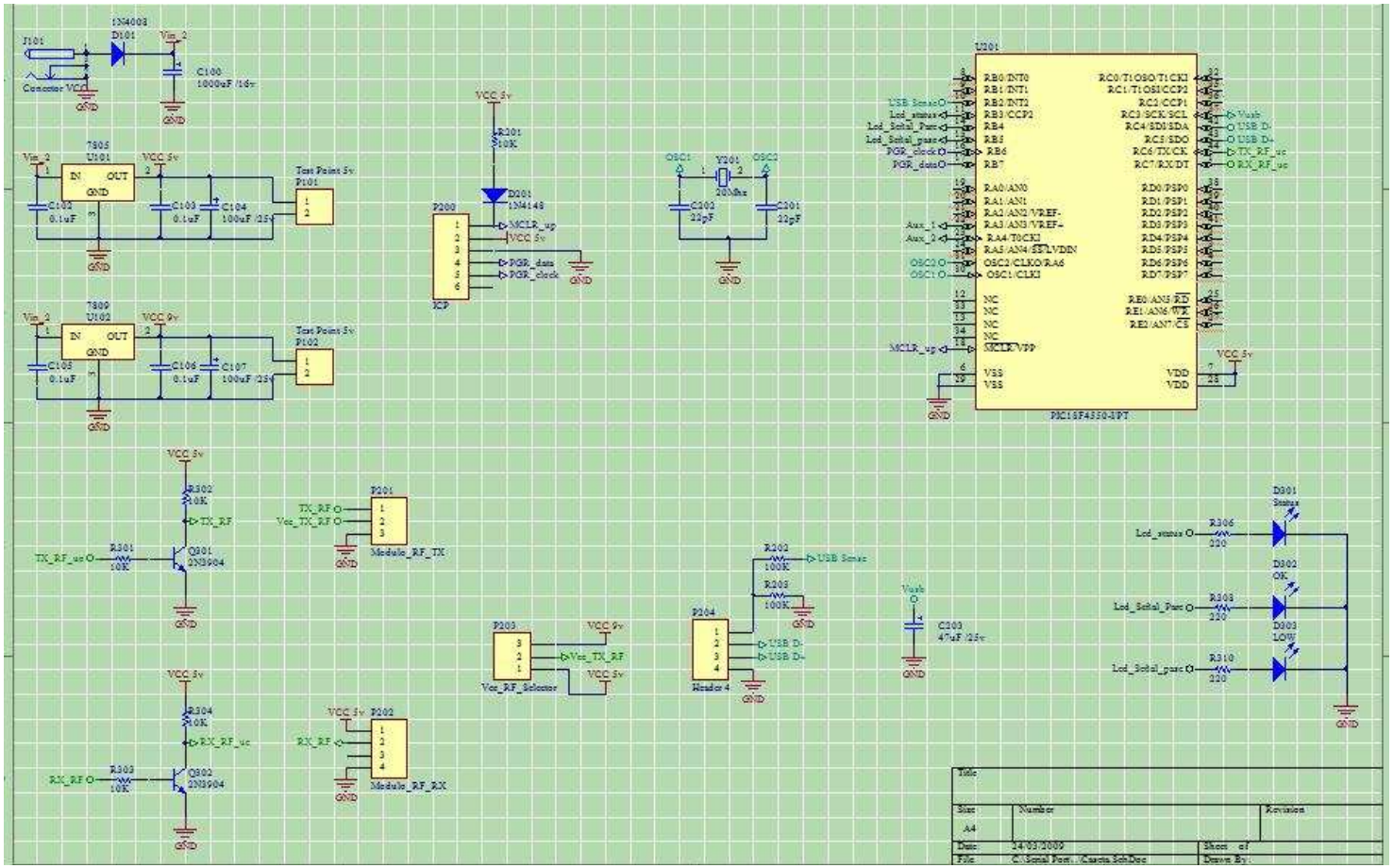
```
{
   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_OFF);
   setup_spi(SPI_SS_DISABLED);
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_256);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DISABLED,0,1);
   setup_comparator(NC_NC_NC_NC);
   setup_vref(FALSE);

   enable_interrupts(int_rda);                    //habilita int. de
recepcion serial
   enable_interrupts(global);                     //habilita
interrupciones globales

   set_tris_b(0b11110001);
   set_tris_c(0b10111111);
   output_bit(LEDsaldo_OK,0);
   output_bit(LEDsaldo_Low,0);
   output_bit(LEDstatus,0);
}
```

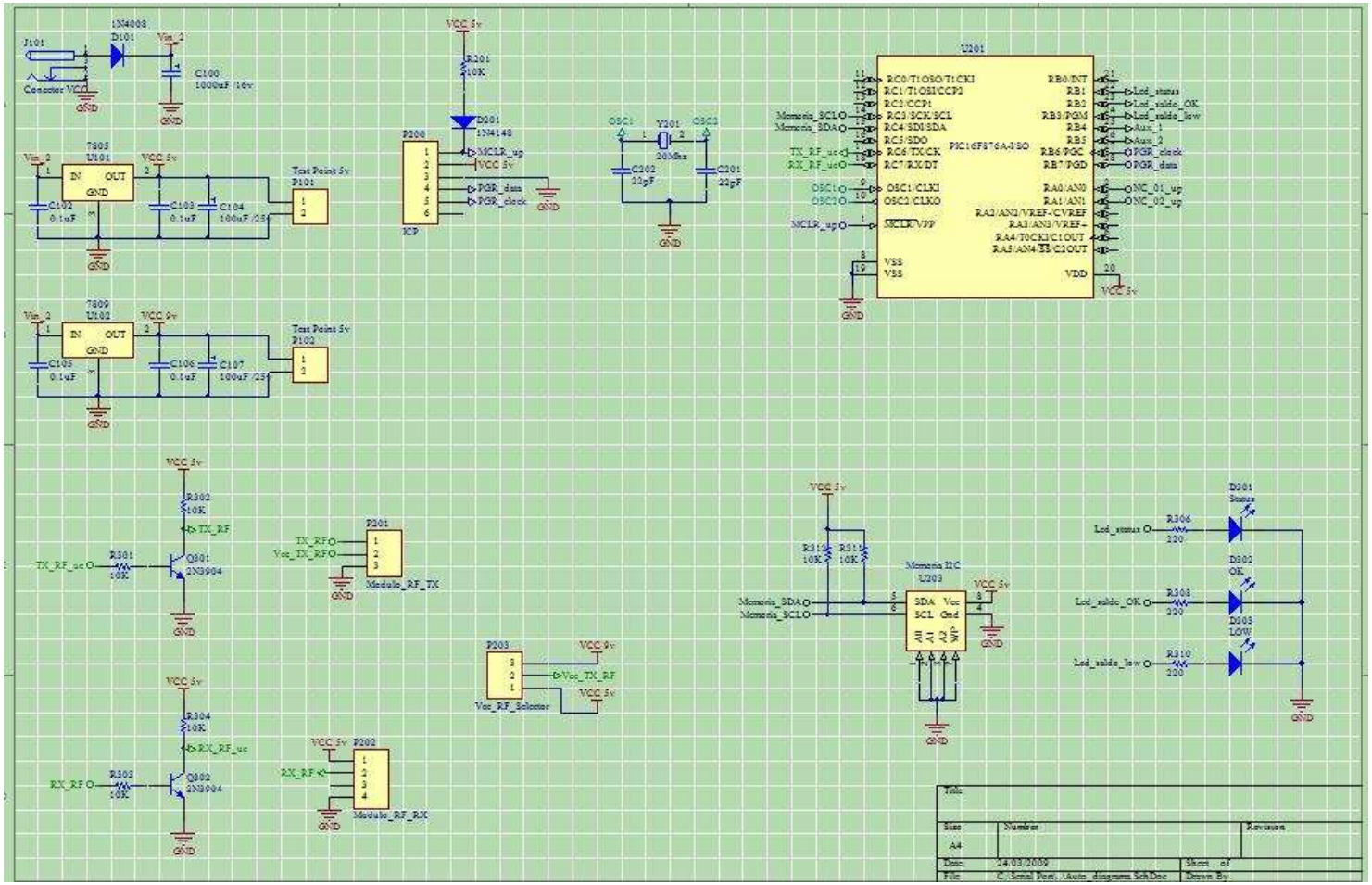
ANEXO C

DIAGRAMA DIFIJO



ANEXO D

DIAGRAMA DIMOVIL



ANEXO E

SOFTWARE PC

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;              //libreria de puertos
using System.Threading;



namespace Comunicacion_serial
{
    public partial class INTRFAZ_USUARIO : Form
    {

        //*************************************************************
        //********   Variables Globales                    ************
        //*************************************************************

        public string data = null;
        public string visor = null;
        public int valor = 0;
        public Boolean DataNueva = false;
        public Boolean cambio = false;
        public string SubData = null;
        public string IDvehiculo = null;
        public string[] PartesTrama;
        public int estadoConexionUSB = 0;
        public int ex = 0;
        public int contadorTimer = 0;
        public int comandoRecibido = 0;
        public int estadoEquipoPeaje = 0;


        //*************************************************************
        //********   Constantes para el manejo de Tramas  ************
        //*************************************************************
        public const int Sol_TarifasPeaje = 1;
        public const int Resp_TarifasPeaje = 2;
        public const int Sol_IDauto = 3;
```

```
public const int Resp_IDauto = 4;
public const int Sol_SaldoAuto = 5;
public const int Resp_SaldoAuto = 6;
public const int Recargar_Saldo = 7;
public const int Cobro_PeajeAuto = 8;
public const int Nuevo_TarifasPeaje = 9;
public const int Sol_IDcaseta =10;
public const int Resp_IDcaseta = 11;
public const int Sol_equipoPeaje = 12;
public const int Resp_equipoPeaje = 13;
public const int nuevoEstadoPeaje = 14;
public const int ACK = 15;


public const int liviano = 1;
public const int pesado = 2;
public const int muy_pesado = 3;
public const int especial = 4;

public const int usb_OK = 1;
public const int usb_vacio = 2;
public const int usb_desconectar = 3;

public const int peajeManual = 2;
public const int peajeAutomatico = 3;

//************************************************************
//********   Declaracion de Funciones o Metodos  ************
//************************************************************

public void EnviarTramaUSB(int TipoTrama, params string[]Datos) {
    string EnviarDatos = null;
    if (Datos.Length > 0)
    {
        for (int i = 0; i < Datos.Length; i++)
        {
            EnviarDatos = EnviarDatos + Datos[i] + "/";
        }
        EnviarDatos = (char)TipoTrama + EnviarDatos;
    }
    else
    {
        EnviarDatos = (char)TipoTrama + "";
    }
    serialPort1.Write(EnviarDatos);
    comandoRecibido = 0;
    contadorTimer = 0;
    timerTimeout.Enabled = true;

}


public void DesactivarBotones()
{
    LeerTarifas.Enabled=false;
```

```
            CargarSaldo.Enabled=false;
            CargarNuevasTarifas.Enabled=false;
            LeerSaldo.Enabled=false;
            botonEstadoPeaje.Enabled=false;
            cobroPeajeManual.Enabled = false;
        }

        public void ActivarBotones()
        {

            CargarSaldo.Enabled = true;
            LeerSaldo.Enabled = true;
            botonEstadoPeaje.Enabled = true;
            //cobroPeajeManual.Enabled = true;
        }


        //*************************************************************
        //********   Programa Principal  *************
        //*************************************************************

        public INTRFAZ_USUARIO()
        {
            InitializeComponent();
            DesactivarBotones();
            BuscarVehiculo.Enabled = false;
        }


        private void serialPort1_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            Thread.Sleep(200);
            this.data = serialPort1.ReadExisting();
            DataNueva = true;

        }


        private void timer1_Tick(object sender, EventArgs e)
        {
            if (DataNueva == true)
            {
                DataNueva = false;
                comandoRecibido=Convert.ToInt16(data[0]);
                labelTimeout.Text = comandoRecibido.ToString();
                switch (comandoRecibido)
                {
                     case Resp_TarifasPeaje:    PartesTrama =
this.data.Split(new char[] { '/' });
                                                textBox2.Text =
PartesTrama[1];         // livianos
                                                textBox3.Text =
PartesTrama[2];         // pesados
```

```
                                                    textBox4.Text =
PartesTrama[3];         // muy pesado
                                                    textBox5.Text =
PartesTrama[4];         // especiales
                                                    break;

                        case Resp_SaldoAuto:        PartesTrama =
this.data.Split(new char[] { '/' });
                                                    label3.Text = "ID  " +
PartesTrama[1];      //Presenta  ID del ehiculo
                                                    label4.Text = "$ " +
PartesTrama[2];        //Presenta Saldo actual del vehiculo
                                                    break;

                            case Resp_IDauto:       PartesTrama =
this.data.Split(new char[] { '/' });
                                                    label1.Text =
PartesTrama[1];  //muestro ID del vehiculo
                                                    IDvehiculo =
PartesTrama[1];
                                                    switch (PartesTrama[2])
//muestro que tipo de auto es
                                                    {
                                                        case "1":   label2.Text
= "Liviano";
                                                                    break;
                                                        case "2":   label2.Text
= "Pesado";
                                                                    break;
                                                        case "3":   label2.Text
= "Muy Pesado";
                                                                    break;
                                                        case "4":   label2.Text
= "Especial";
                                                                    break;
                                                        default:    label2.Text
= "Desconocido";
                                                                    break;
                                                    }
                                                    ActivarBotones();
                                                    break;
                            case Resp_IDcaseta:     PartesTrama =
this.data.Split(new char[] { '/' });
                                                    label9.Text =
PartesTrama[1];
                                                    break;
                        case Resp_equipoPeaje:      PartesTrama =
this.data.Split(new char[] { '/' });
                                                    switch (PartesTrama[1])
//muestro que tipo de auto es
                                                    {
                                                        case "2":
labelEstadoPeaje.Text = "Manual";

estadoEquipoPeaje = peajeManual;
```

```
botonEstadoPeaje.Text = "Cambiar a Automático";

cobroPeajeManual.Enabled = true;
                                                                    break;
                                                            case "3":
labelEstadoPeaje.Text = "Automatico";

estadoEquipoPeaje = peajeAutomatico;

botonEstadoPeaje.Text = "Cambiar a Manual";

cobroPeajeManual.Enabled = false;
                                                                    break;
                                                            default:
labelEstadoPeaje.Text = "Desconocido";

estadoEquipoPeaje = 0;

botonEstadoPeaje.Text = "Leer Estado";
                                                                    break;
                                                        }
                                                        break;
                                        default:        label1.Text = "  ---";
                                                        break;
                    }
                    if (comandoRecibido>=0)
                    {
                        timerTimeout.Enabled = false;
                        this.panel1.BackColor = System.Drawing.Color.Green;
                        labelTimeout.Text = "      OK";


                    }
                    this.data = null;
                }

            }


             private void Form1_Load(object sender, EventArgs e)
             {

                 string[] ports = SerialPort.GetPortNames();
                 if (ports != null && ports.Length > 0)
                 {
                     DataTable dt = new DataTable();
                     dt.Columns.Add("nombre", typeof(string));
                     for (int i = 0; i < ports.Length; i++)
                     {
                         DataRow dr = dt.NewRow();
                         dr["nombre"] = ports[i];
                         dt.Rows.Add(dr);
                     }
                     cmbPorts.DataSource = dt;
```

```
                cmbPorts.DisplayMember = "nombre";
                cmbPorts.ValueMember = "nombre";
                estadoConexionUSB = usb_OK;
            }
            else
            {
                conectarEquipo.Enabled = false;
                label9.Text = "No hay Conexion";
            }
        }


        //************************************************************
        //********   Botones  *************
        //************************************************************



        private void LeerTarifas_Click(object sender, EventArgs e)
        {
            EnviarTramaUSB(Sol_TarifasPeaje);
            CargarNuevasTarifas.Enabled = true;
        }

        private void CargarSaldo_Click(object sender, EventArgs e)
        {
            label3.Text = "";       //Borro Saldo Actual
            label4.Text = "";
            EnviarTramaUSB(Recargar_Saldo, textBox1.Text);
        }

        private void CargarNuevasTarifas_Click(object sender, EventArgs e)
        {

EnviarTramaUSB(Nuevo_TarifasPeaje,textBox2.Text,textBox3.Text,textBox4.Text
,textBox5.Text);
        }


        private void BuscarVehiculo_Click(object sender, EventArgs e)
        {
            label1.Text = "";
            label2.Text = "";
            EnviarTramaUSB(Sol_IDauto);

        }

        private void conectarEquipo_Click(object sender, EventArgs e)
        {
            if (estadoConexionUSB == usb_OK)
            {
                serialPort1.PortName = cmbPorts.SelectedValue.ToString();
                if (!serialPort1.IsOpen)
                {
                    try
```

```
                {
                    serialPort1.Open();
                    estadoConexionUSB = usb_desconectar;
                    conectarEquipo.Text = "Desconectar";
                    EnviarTramaUSB(Sol_IDcaseta);
                    BuscarVehiculo.Enabled = true;
                    LeerTarifas.Enabled = true;
                    LeerTarifas.Enabled = true;
                    CargarNuevasTarifas.Enabled = true;
                    botonEstadoPeaje.Enabled = true;



                }
                catch (System.Exception ex)
                {
                    MessageBox.Show(ex.ToString());
                }
            }
            else
            {
                MessageBox.Show("puerto ya esta abierto");
            }
        }
        else
        {
            serialPort1.Close();
            estadoConexionUSB = usb_OK;
            conectarEquipo.Text = "Conectar";
            DesactivarBotones();
            BuscarVehiculo.Enabled = false;

        }

    }

    private void cobroPeajeManual_Click(object sender, EventArgs e)
    {
        EnviarTramaUSB(Cobro_PeajeAuto);
    }

    private void LeerSaldo_Click(object sender, EventArgs e)
    {
        label3.Text = "";
        label4.Text = "";
        EnviarTramaUSB(Sol_SaldoAuto);

    }

    private void timerTimeout_Tick(object sender, EventArgs e)
    {
        labelTimeout.Text = "Processing";
        ++contadorTimer;
        labelTimeoutContador.Text = contadorTimer.ToString();
        cambio = !cambio;
```

```
            if (cambio)
            {
                this.panel1.BackColor = System.Drawing.Color.Orange;
            }
            else
            {
                this.panel1.BackColor = System.Drawing.Color.White;
            }

            if (contadorTimer >= 15)
            {
                timerTimeout.Enabled = false;
                DesactivarBotones();
                this.panel1.BackColor = System.Drawing.Color.Red;
                labelTimeout.Text = "Timeout";
            }
        }

        private void botonEstadoPeaje_Click(object sender, EventArgs e)
        {
            switch (estadoEquipoPeaje)
            {
                case 0: EnviarTramaUSB(Sol_equipoPeaje);
                        break;
                case 2:
EnviarTramaUSB(nuevoEstadoPeaje,peajeAutomatico.ToString());
                        break;
                case 3: EnviarTramaUSB(nuevoEstadoPeaje,
peajeManual.ToString());
                        break;
            }
        }

        private void groupBox6_Enter(object sender, EventArgs e)
        {

        }

        private void label1_Click(object sender, EventArgs e)
        {

        }

        private void label10_Click(object sender, EventArgs e)
        {

        }

        private void groupBox1_Enter(object sender, EventArgs e)
        {

        }

        private void groupBox8_Enter(object sender, EventArgs e)
        {
```

```
        }

        private void groupBox3_Enter(object sender, EventArgs e)
        {

        }

        private void button1_Click(object sender, EventArgs e)
        {

        }

        private void label4_Click(object sender, EventArgs e)
        {

        }


        private void label3_Click(object sender, EventArgs e)
        {

        }


    }
}
```

ANEXO F

INTERFAZ DE USUARIO

El programa “interfaz de usuario” es desarrollado en el lenguaje de programación visual C#. Este interfaz tiene 4 Menús los cuales se detallará más adelante.

La conexión entre la computadora y el DIFIJO se realiza con un cable USB. Esta conexión emula un comunicación serial. Cuando conectamos el dispositivo debemos chequear que puerto **com** se crea, esto se revisa en la ventana de *administración de dispositivos* del Sistema Operativo de la PC, la cual se aprecia en la figura 1, en este caso se crea el puerto **com5.**

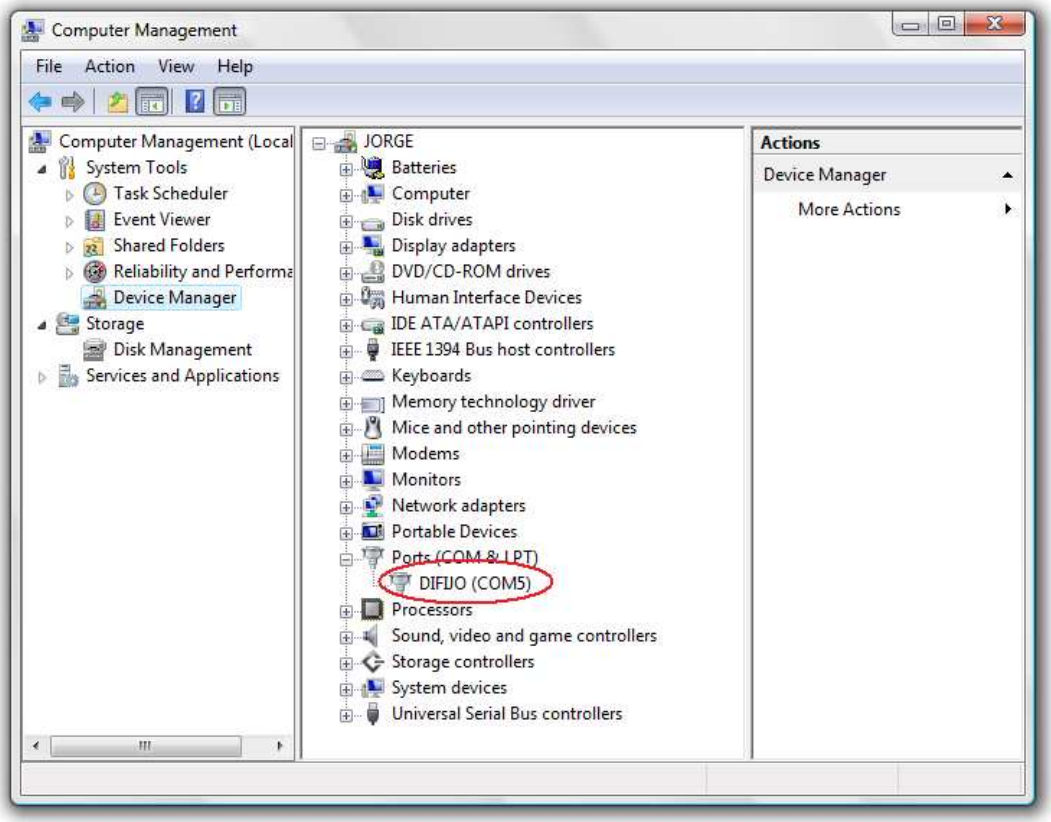


Figura 1 Ventana de Administración de Dispositivos

En la ventana principal de la “Interfaz de Usuario” se debe escoger el puerto de comunicación que se creó (com5) y pulsamos la pestaña **conectar** (Figura 2).

Figura 2 Pantalla inicial Interfaz De Usuario

Al conectarse con el DIFIJO en el campo **Estado Conexión** aparece la palabra **OK** y en **Equipo Peaje** aparece el nombre del DIFIJO, en este caso **CASETA 1** (Figura 3).

Figura 3 Conexión Satisfactoria PC - DIFIJO

Para ver los valores de peaje desplegamos la ventana **Tarifas** (Figura 4). En este campo pulsamos la pestaña **Leer Tarifas** y se despliega los valores de cobro para los distintos tipos de vehículos.

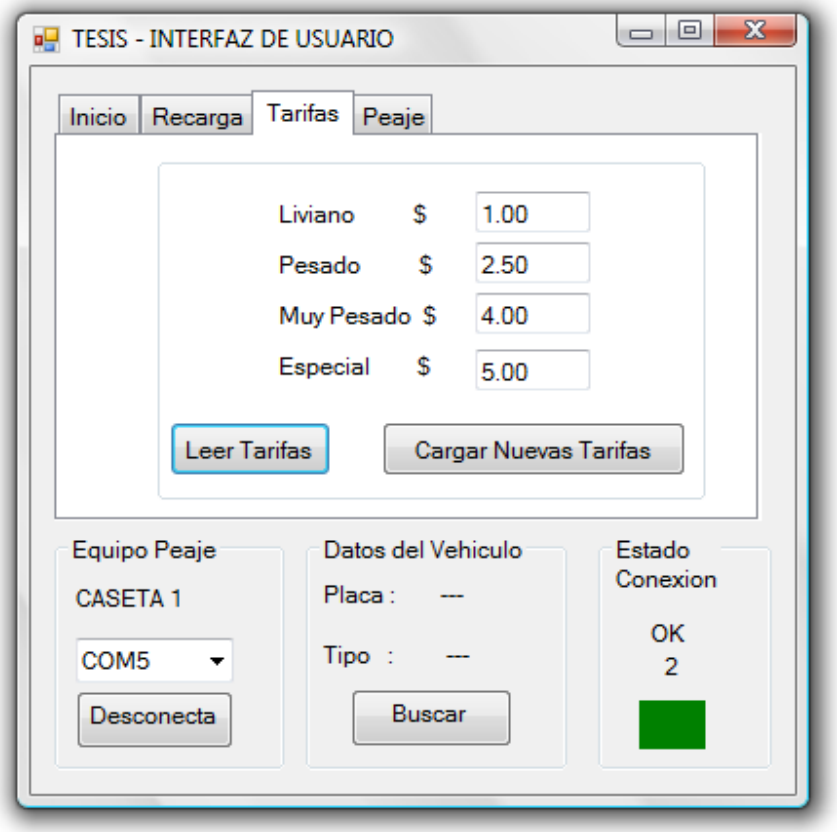


Figura 4 Leer tarifas del DIFIJO

Para modificar los valores de cobro, colocamos en los campos de los diferentes tipos de vehículos las nuevas tarifas y pulsamos la pestaña **Cargar Nuevas Tarifas**, esos nuevos valores se guardan en el DIFIJO.

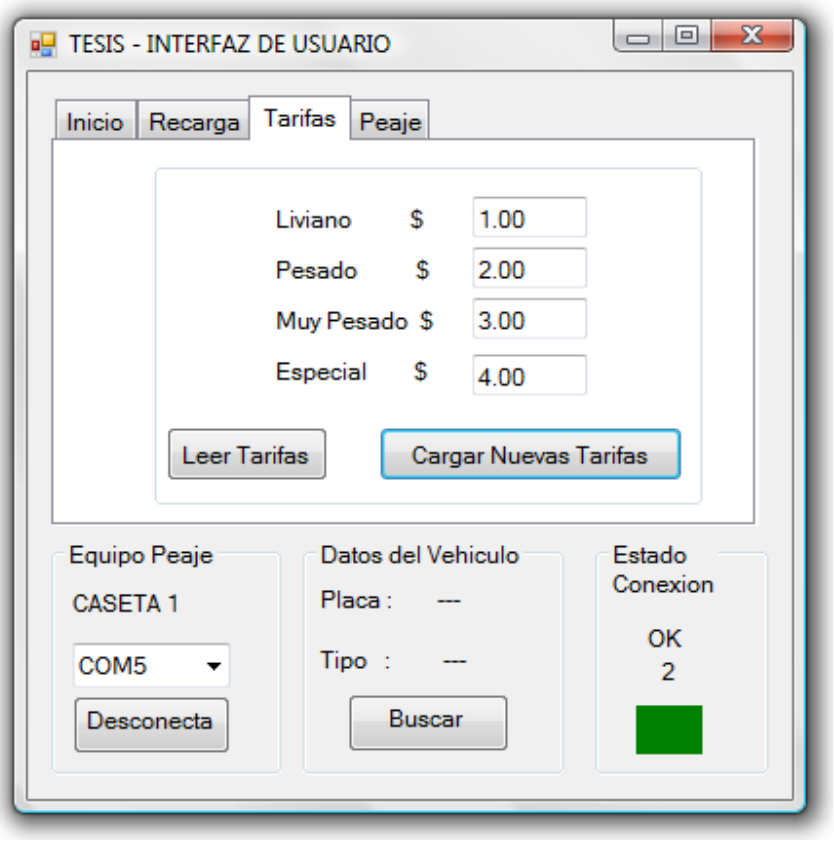


Figura 5 Cargar nuevas tarifas al DIFIJO

El DIFIJO puede estar configurado para el cobro MANUAL o AUTOMÁTICO, ese estado se lo puede observar en la ventana **Peaje.**

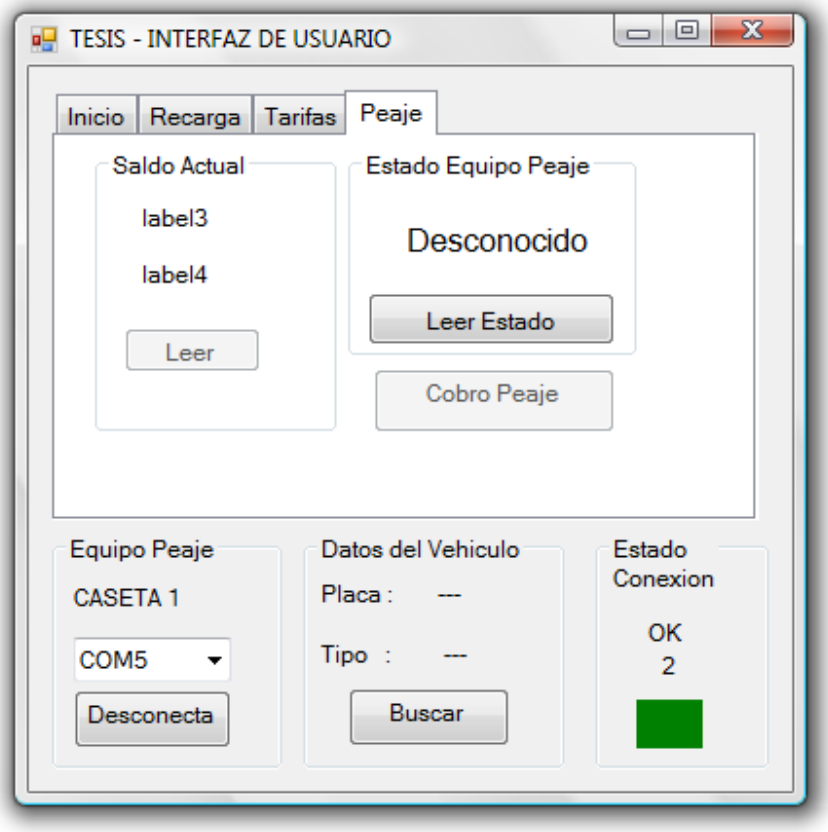


Figura 6 Opción Peaje de la barra de Menú

Al pulsar la pestaña **Leer Estado** se despliega el estado de cobro, en este caso el DIFIJO está configurado para cobro MANUAL (Figura 7).

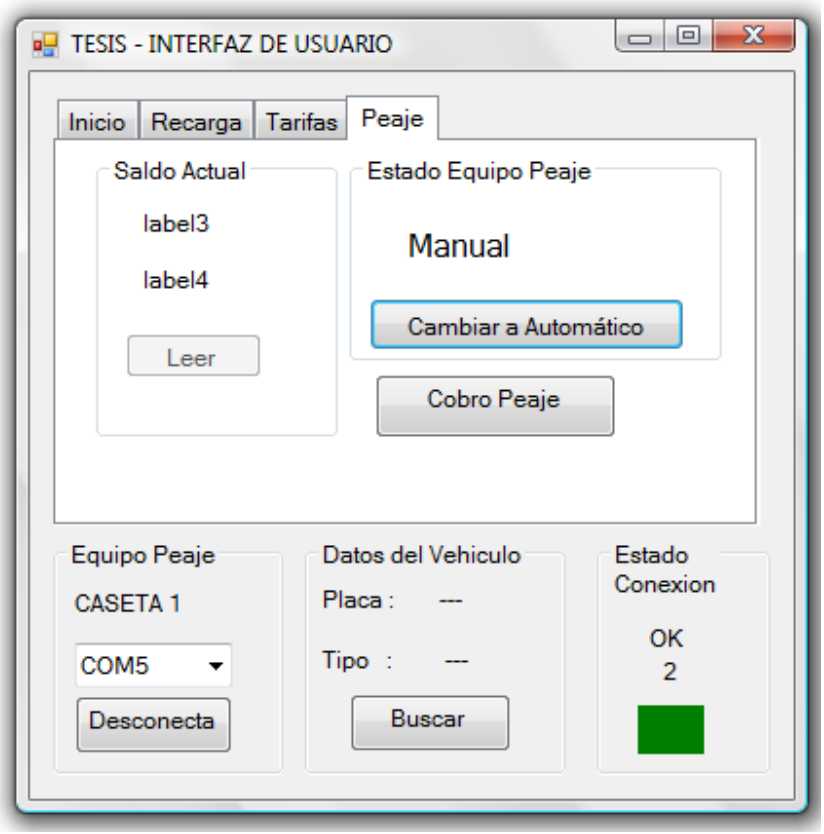


Figura 7 Forma Manual

En la forma MANUAL cada vez que se vaya a cobrar el peaje se debe pulsar la pestaña **Cobro Peaje**, en el campo **Saldo Actual** se despliega el ID del vehículo al cual se le cobra y el saldo que queda disponible.

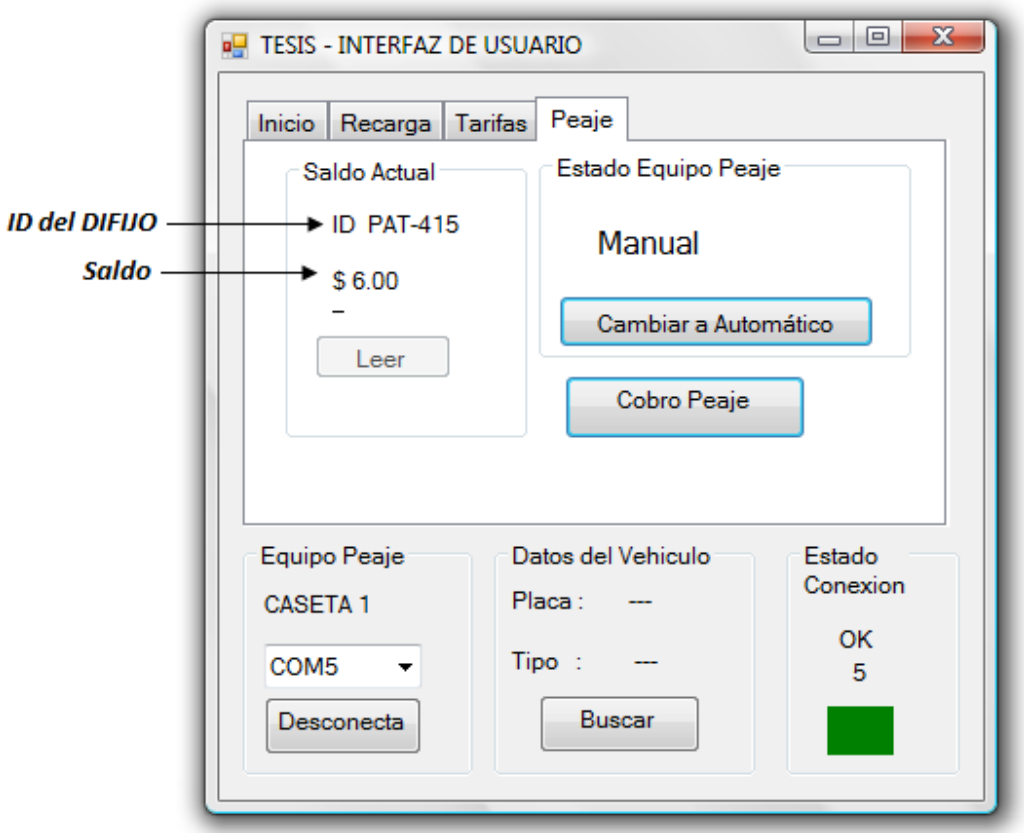


Figura 8 Cobro Peaje Manual

En la forma automática, el DIFIJO cobra automáticamente el peaje, espera 7 segundos y nuevamente envía la trama de Broadcast en busca de autos que tengan incorporado el DIMOVIL para repetir el proceso de cobro. Para cambiar a este estado pulsamos la pestaña **Cambiar a Automático** y en el campo **Estado Equipo Peaje** aparece la palabra **Automático**, en **Datos del Vehículo** se observa la Placa (ID) y tipo de vehículo al que se le realizó el cobro.

Para regresar a la forma manual, pulsamos la pestaña **Cambiar a Manual.**

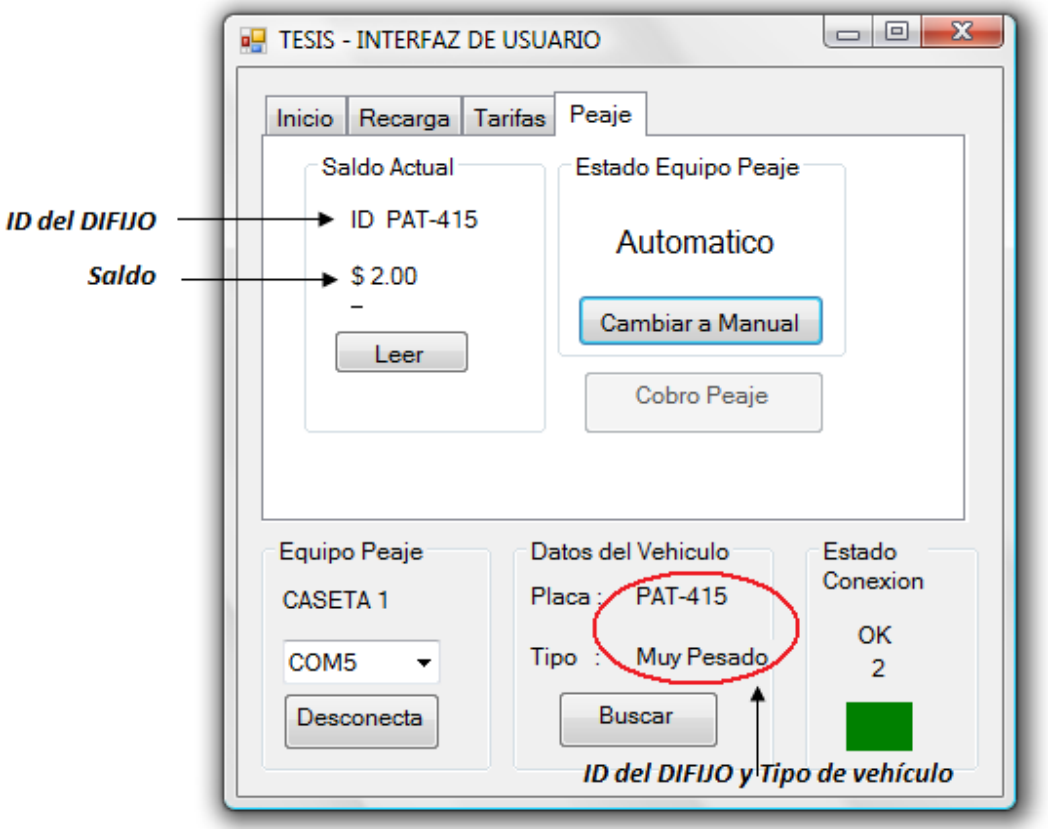


Figura 9 Cobro Peaje Automático

Para realizar una recarga se despliega la ventana **Recarga** (Figura 10), al pulsar la pestaña **Buscar** el DIFIJO localiza al vehículo, los datos de este se observa en el campo **Datos del Vehículo**, colocamos el valor a recargarse (para este caso se va a realizar una recarga de 50 dólares) en el campo **Saldo a Cargar** y al pulsar la pestaña **Recargar**, el nuevo valor se sumará al saldo que disponga en ese instante ese vehículo.

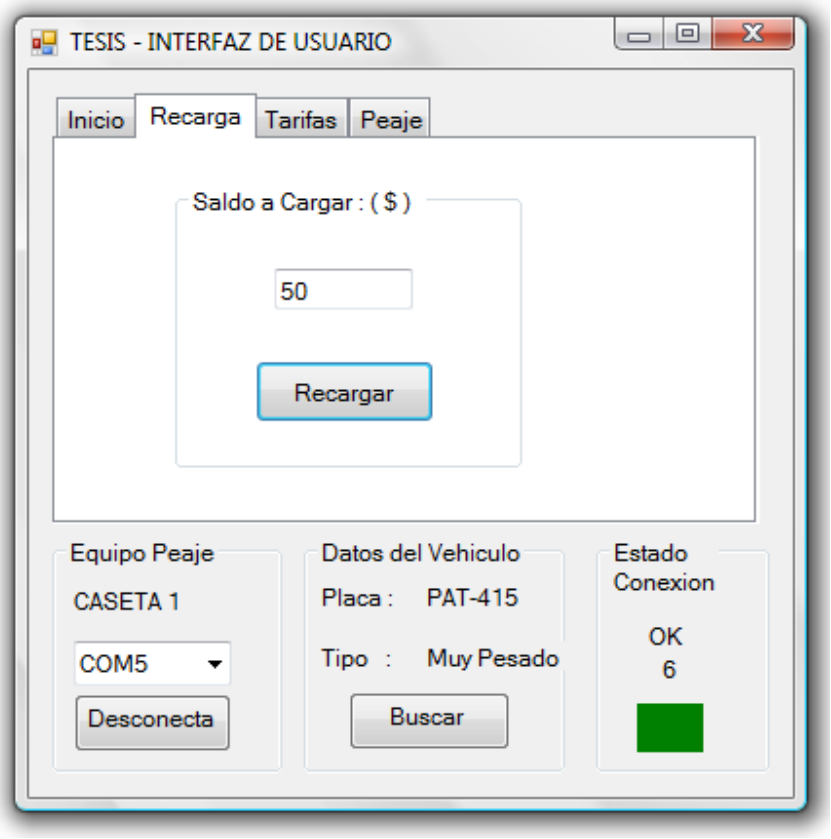


Figura 10 Recarga de Saldo