Faculdade de Engenharia da Universidade do Porto

FEUP

Sistema de monitorização de dados baseado em microprocessador

Ricardo Jorge Ribeiro Moreira

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Electrotécnica e de Computadores
Major Automação

Orientador: Prof. Dr. Adriano Carvalho

10 de Junho de 2010

Resumo

No curso de Engenharia Electrotécnica e de computadores, vertente de automação, ministrado pela Faculdade de Engenharia da Universidade do Porto são leccionadas matérias diversas, entre elas sistemas baseados em microprocessadores. Na tomada de decisão para a escolha do trabalho a realizar no âmbito da dissertação, considerou-se um factor primordial o aprofundar de conhecimentos nesta área dado o forte desenvolvimento das tecnologias a nível de comunicações.

Este projecto tem como objectivo o desenvolvimento e implementação de um contador de tempo de jogo de bilhar, transferindo a sua informação para um computador, não havendo ligação directa por cabo.

A troca de informação entre o contador e o computador não poderá ser cablada e terá que permitir comunicação de vários contadores e o computador, o que obrigará a um sistema intermédio que servirá de interface.

O trabalho realizado no âmbito desta dissertação aborda as diversas fases de projecto, implicando numa primeira fase o estudo comparativo entre microcontroladores e FPGAs e também o estudo comparativo entre várias alternativas para comunicação *wireless* de modo a estabelecer um quadro de trabalho apropriado. A segunda fase do trabalho consiste no desenvolvimento do projecto, cumprindo com os requisitos iniciais

Abstract

In the course of Electrical Engineering and Computer Automation, given by the Faculty of Engineering, University of Porto are taught different subjects, including systems based on microprocessors. To choose the work in the dissertation, the improval of knowledge in this area thus the strong development of technologies with communications was considered a major factor.

This project aims at developing and implementing a billiard time counter, sending its information to a computer, without direct cable connection.

The exchange of information between the counter and the computer can not be wired and will have to allow communication of multiple counters and the computer, which will force the system to have one communication interface.

The work done in this thesis addresses the different steps of design: initially the comparative study of microcontrollers and FPGAs as well as the comparative study of various alternatives for wireless communication in order to establish a framework for appropriate work was made. The second step of this work is the development of the project, fulfilling the initial requirements.

Agradecimentos

Ao longo do meu trabalho, alguns foram os que contribuíram com ajuda e motivação, sem as quais o presente trabalho não teria sido possível.

A todos aqueles que, de alguma forma contribuíram para a realização deste trabalho, deixo aqui os meus sinceros agradecimentos.

Índice

Lista de figuras

Lista de tabelas

Abreviaturas e Símbolos

Lista de abreviaturas (ordenadas por ordem alfabética)

A/D	*Analog To Digital*
ACK	*Acknowledge*
AES	*Advanced Encryption Standart*
AODV	*Ad-hoc On-demand Distance Vector*
ASIC	*Application-Specific Integrated Circuit*
BRG	*BaudRate Generator*
CAN	*Controller Area Network*
CPU	*Central Processing Unit*
D/A	*Digital To Analog*
DSS	Sequência Directa
E/S	Entrada / Saída
EPROM	*Erasable Programmable Read-Only Memory*
EEPROM	*Electrically-Erasable Programmable Read-Only Memory*
FFD	*Full Function Device*
F_{OSC}	Frequência do Oscilador
FPGA	*Field Programmable Gate Array*
HDL	Linguagem Descritiva de Hardware
I/O	*Input / Output*
I^2C	*Inter-Intergrated Circuit*
ICSP	*In-Circuit Serial Programming*
IEEE	*Institute of Electrical and Eletronics Engineers*
ISM	*Industrial, Scientific and Medical*
LSB	*Less Significant Bit*
LUT	*Look Up Table*
MAC	*Media Access Control*
MSSP	*Master Synchronous Serial Port*

NRZ	*Non-Return-To-Zero*
NWK	*Network*
OTP	*One-Time-Programmable*
P2P	*Peer-To-Peer*
PAN	*Personal Area Network*
PC	*Personal Computer*
PCB	*Printed Circuit Board*
PDA	*Personal Digital Assistants*
PHY	*Physical*
PLD	*Programmable Logic Devices*
PROM	*Programmable Read Only Memory*
PSP	*Parallel Slave Port*
PWM	*Pulse-Width Modulation*
RAM	*Random Access Memory*
RFD	*Reduced Function Device*
ROM	*Read Only Memory*
Rx	*Receive*
SCK	*Serial Clock*
SCL	*Serial Clock*
SDA	*Serial Data*
SDI	*Serial Data Input*
SDO	*Serial Data Output*
SFR	*Special Function Registers*
SPI	*Serial Peripheral Interface*
SS	*Slave Selection*
TMR	*Timer*
Tx	*Transmit*
UART	*Universal Asynchronous Receiver Transmitter*
USART	*Universal Synchronous Asynchronous Receiver Transmitter*
WLAN	*Wireless Local Area Network*
WMAN	*Wireless Metropolitan Area Network*
WPAN	*Wireless Personal Area Network*
WWAN	*Wireless Wide Area Network*

Capítulo 1

Introdução

Esta dissertação, tal como o nome indica, consiste no desenvolvimento de um projecto de monitorização e controlo baseado na tecnologia de microprocessadores.

Neste capítulo será discutida a evolução tecnológica que antecede este projecto, quais os motivos que despertam interesse numa nova solução, em que aspecto este projecto poderá contribuir para um novo produto disponível no mercado e finalmente será explicada a organização desta dissertação.

1.1 - Evolução Tecnológica

Decorria o ano de 1987, quando as oportunidades e necessidades do mercado nacional criaram espaço para o desenvolvimento e lançamento do primeiro contador electrónico de tempo de jogo de bilhar. Tratava-se do modelo CMM2000 da marca RPM. Este modelo tinha um conjunto de cinco displays, sensores de pressão para três bolas e dois botões que permitiam a escolha da tarifa. O processamento da informação era então suportado em lógica discreta.

Em meados de 1994, a Contelec, uma empresa recém criada, apresenta ao mercado um novo conceito de contador electrónico de tempo de bilhar. Tratava-se do modelo JP2002. As inovações, quer estruturais quer internas, eram significativas. A partir desse momento passou a haver armazenamento das 16 bolas e o processamento era feito através de um microcontrolador.

Uma nova revolução surge em 2004, quando é apresentado ao mercado uma vasta gama de produtos que trouxeram para a realidade o conceito de sala de jogos informatizada.

Este novo sistema, permite interligar vários contadores de tempo de bilhar, placas controladoras de luzes de bilhar, painéis publicitários e placas de controlo de máquinas recreativas ao computador, através de uma rede cablada baseada no protocolo RS485, sendo

possível monitorizar e controlar todo o salão de jogos através de uma aplicação especialmente desenvolvida denominada "Contelec Live", em que é possível controlar as luzes dos bilhares independentes, visualizar o estado de cada bilhar, se está em jogo ou não e caso esteja quanto é que está a marcar, verificar quantas moedas estão presentes em cada máquina recreativa, efectuar consultas por datas e equipamentos, entre outras opções.

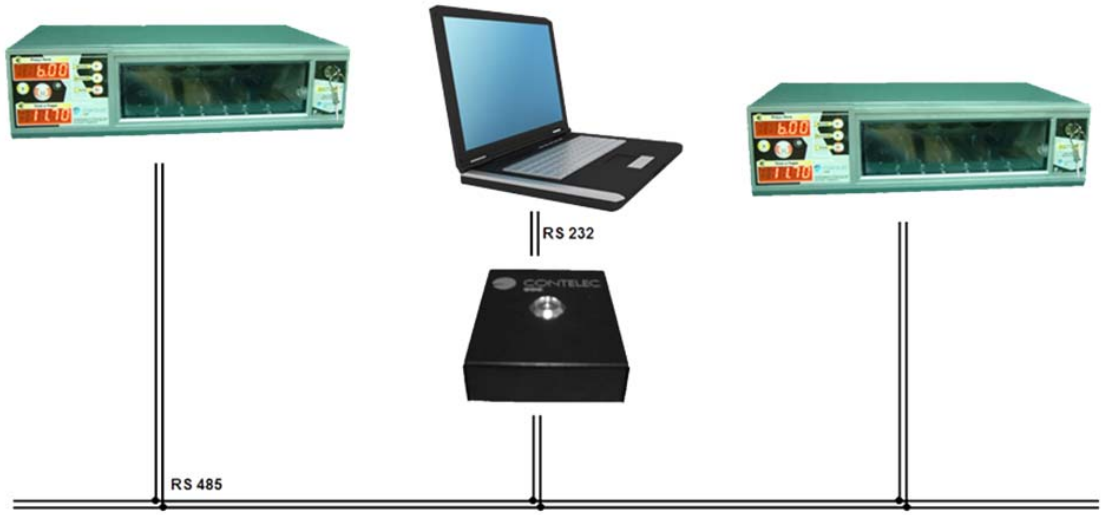


Figura 1.1 – Interligações do sistema existente

1.2 - Importância de uma nova solução

A rede interna que liga os contadores de tempo de bilhar ao computador era gerida por uma interface que servia de intérprete entre os mesmos e o computador. Deste modo era possível ligar vários contadores de tempo de bilhar em simultâneo. O único inconveniente deste tipo de instalação é a obrigação da instalação de um cabo de rede que interligasse cada contador, o que obrigaria a que os salões de jogos preparassem a instalação previamente. Uma vez que tal instalação requer demasiados recursos, surge então a possibilidade de desenvolver um novo produto que permita as mesmas funcionalidades, sem a desvantagem de ser dependente de cablagem.

A única solução possível para contornar este problema seria optar por uma solução de comunicação sem fios. Abre-se então a porta para o estudo de alternativas existentes no mercado, tipos de tecnologias sem fios e possíveis alternativas aos microcontroladores.

1.3 - Contribuição para uma nova solução

Desenvolver um projecto que cumprisse todas as especificações do projecto actual é uma tarefa exigente a nível de tempo, uma vez que o projecto existente demorou dois anos a ser desenvolvido. Consequentemente os objectivos foram ajustados de modo a que seja possível a

sua finalização. Os objectivos básicos serão os mesmos, sendo necessária a criação de uma interface que servirá de intermediário entre os contadores de tempo de bilhar e o computador e terá que permitir a ligação de até vinte contadores de tempo de bilhar em simultâneo. Cada contador de tempo de bilhar terá que guardar os registos dos totais, permitir programação de tarifa (através de três interruptores de pressão), detectar se todas as bolas se encontram devidamente guardadas (com o auxílio de quatro sensores do tipo *microswitch*) e terá cinco displays para mostrar a tarifa e outros cinco para o total a pagar. O circuito PCB terá que ter as mesmas dimensões e o mesmo posicionamento dos displays e interruptores de pressão que o projecto actual, de modo a viabilizar economicamente o projecto.

Futuramente, o projecto irá ser continuado, de modo a que todas as funcionalidades do projecto existente estejam presentes.

1.4 - Especificações da nova solução

Como foi dito anteriormente, o projecto terá que ser ajustado ao tempo disponível. Desse modo, irão ser desenvolvidos dois dispositivos distintos: interface de comunicações e contador de tempo de bilhar e irá também ser desenvolvido um software de monitorização do salão de jogos.

O interface de comunicações será um descodificador de comunicação sem fios para uma comunicação cablada e vice-versa, que interligue directamente ao computador e através da rede sem fios, comunique com todos os contadores de tempo de bilhar presentes.

Relativamente a cada contador de tempo de bilhar, este terá que armazenar os registos totais, permitir programação de tarifa, detectar se todas as bolas se encontram devidamente armazenadas, a tarifa e total a pagar terão que ser visíveis através de 5 displays cada. O PCB do contador de tempo de bilhar terá que ter um determinado tamanho, assim como a posição dos displays e teclas de programação terão que se encontrar em determinadas coordenadas.

A nível de comunicações o sistema tem que ser capaz de interligar até 10 contadores de tempo de bilhar.

A aplicação desenvolvida terá que ser capaz de efectuar uma leitura e/ou escrita dos valores da EEPROM de cada contador de tempo de bilhar, permitir a monitorização de recepção e envio de tramas numa *form* que poderá estar visível ou oculta, armazenar todos os pagamentos efectuados numa base de dados assim como permitir a consulta desses pagamentos por data e por contador de tempo de bilhar.

Desenvolver um projecto que cumprisse todas as especificações do projecto actual é uma tarefa exigente a nível de tempo, uma vez que o projecto existente demorou dois anos a ser desenvolvido. Consequentemente os objectivos foram ajustados de modo a que seja possível a

1.5 - Organização da dissertação

Esta dissertação encontra-se estruturada em cinco capítulos, bibliografia e referências e um anexo, da seguinte forma:

- Capítulo 1 – Apresenta uma introdução à dissertação, apresenta a evolução tecnológica do ramo em que se insere, descreve a importância de uma nova solução assim como a sua contribuição com este projecto e descreve também o modo como esta dissertação é apresentada
- Capítulo 2 – Apresenta as tecnologias de hardware existentes, nomeadamente FPGAs e microcontroladores e também apresenta uma tecnologia sem fio emergente (ZigBee) e duas alternativas simplistas (MiWi e MiWi P2P).
- Capítulo 3 – Este capítulo está dividido em dois subcapítulos bem distintos: arquitectura de sistema e projecto e desenvolvimento. No subcapítulo da arquitectura de sistema são efectuadas análises comparativas das tecnologias de hardware e de rede sem fios, concluindo que hardware e software é mais adequado ao projecto, é analisado o microcontrolador escolhido, assim como o módulo de rádio escolhido para a comunicação sem fios. No subcapítulo seguinte, é apresentado todo o processo de esquematização, desenho e produção do circuito de PCB, assim como a sua montagem. Apresentam-se os algoritmos utilizados, assim como a estrutura dos dados da placa de interface e da da placa desenvolvida para o contador de tempo de bilhar. É também apresentada a aplicação desenvolvida em Visual Basic para monitorização e controlo.
- Capítulo 4 – Os resultados obtidos são apresentados e discutidos neste capítulo.
- Capítulo 5 – Apresenta as conclusões a que se chegou com o desenvolvimento deste trabalho. São também incluídas algumas sugestões para a futura continuação do desenvolvimento do projecto, a partir da análise de resultados atingidos.

Capítulo 2

Análise e Especificação

De acordo com as especificações descritas no capítulo anterior, verifica-se que o sistema consiste numa arquitectura distribuída, sendo constituída por um nó mestre e diversos nós escravos, em que o mestre irá criar e gerir a rede sem fios e ser o elemento intermediário entre os escravos e entre o computador.

Uma arquitectura distribuída implica o desenvolvimento de hardware e software. De modo a efectuar a escolha apropriada face à oferta de hardware existente, é indispensável efectuar uma análise das tecnologias existentes. Uma vez que as redes sem fios são as tecnologias emergentes, existe um conjunto variado de soluções de comunicação, obrigando a que haja uma análise de alternativas existentes.

Os elementos fundamentais a desenvolver para implementar uma arquitectura deste tipo são o nó *master*, que desempenha uma função de controlo de rede e os nós *slaves,* uma vez que todos os nós necessitam de implementar um protocolo de *stack* de comunicação sem fios.

De modo a efectuar as análises anteriormente referidas, este capítulo foi dividido em duas partes distintas: tecnologias de hardware e tecnologias de rede sem fio. Na primeira parte, tal como o nome sugere, apresenta-se a história da tecnologia, as suas aplicações, a sua arquitectura e o seu modo de programação. Na segunda parte, apresenta-se detalhadamente uma tecnologia de comunicação com um elevado potencial e duas alternativas simplistas, possivelmente mais atractivas para esta arquitectura.

2.1 - Tecnologia de Hardware

Actualmente a tecnologia evolui a um ritmo bastante elevado, tornando-se obrigatório um estudo prévio das tecnologias existentes antes do início de qualquer projecto.

A electrónica digital tem evoluido desde simples portas lógicas, passando pelos transistores até aos microprocessadores e circuitos lógicos programáveis.

A dependência de componentes externas, tais como memória RAM, memória de dados, etc., limita a utilização de microprocessadores em pequenas aplicações, onde o tamanho físico se torna importante. Para contornar esse problema, surgiram então os microcontroladores, em que o caminho para a evolução tem sido no sentido da integração de periféricos no próprio microcontrolador, de modo a permitir uma integração rápida dos diversos dispositivos que consituem um sistema.

Relativamente aos circuitos lógicos programáveis, existem várias tecnologias disponiveis, sendo que os FPGAs são a mais emergente. A sua evolução tem consistido no aumento significativo do número de portas programáveis, em que o primeiro FPGA continha 64 portas programáveis e actualmente existem dispositivos com milhões de portas programáveis, tornando-se deste modo verdadeiramente eficazes no processamento em paralelo.

Desta análise, exclui-se os microprocessadores do tipo DSP uma vez que, tal como o nome indica, são microprocessadores especializados em processamento digital de sinal, bastante utilizados no processamento de sinais de audio e video pela sua velocidade de funcionamento bastante elevada, quando comparada com microcontroladores convencionais, uma vez que à partida o sistema não requer elevado processamento de sinal.

Este capítulo aborda a história, as aplicações, a arquitectura e o modo de programação, quer de fpgas como de microcontroladores.

2.1.1 - FPGA

Um FPGA é um dispositivo semicondutor que é amplamente utilizado no processamento de informação digital. Foi criado pela Xilinx como sendo um dispositivo que poderia ser programado de acordo com as aplicações do programador. Os FPGAs são constituídos por três tipos de componentes bem distintos: blocos lógicos programáveis, matriz de interligações e pelos blocos de entrada/saída.

Os blocos lógicos podem ser programados de modo a executarem funções de portas lógicas básicas, tais como AND ou XOR ou funções combinacionais mais complexas tais como descodificadores ou funções matemáticas. Na maior parte dos FPGAs, os blocos lógicos contém também elementos de memória, que podem ser simples flip-flops ou blocos de memória mais complexos. [4]

Figura 2.1 - Exemplo de um FPGA

2.1.1.1 - História

«A indústria de FPGAs derivou das memórias programáveis apenas de leitura (PROM) e dos dispositivos lógicos programáveis (PLD). Ambos os PROMs e os PLDs tinham a possibilidade de serem programados na fábrica ou no local, contudo era necessária uma ligação física entre as portas lógicas.

Em 1985, os fundadores da Xilinx, Ross Freeman e Bernard Vonderschmitt, inventaram o primeiro FPGA comercialmente viável - o XC2064. O XC2064 tinha portas programáveis e interligações programáveis entre as portas – era o início de uma nova tecnologia e de um novo mercado. O XC2064 ostentava uns meros 64 blocos lógicos configuráveis (CLBs), com duas tabelas de três entradas de pesquisa cada (LUTs). Mais de 20 anos depois, Freeman entrou no corredor da fama pela sua invenção.

Na década de 1980 o Departamento de Guerra Naval financiou uma estudo proposto por Steve Casselman para desenvolver um computador que iria implementar 600.000 portas reprogramáveis. O estudo foi bem sucedido e uma patente relacionada com o sistema foi registada em 1992.

A Xilinx continuou sem concorrência e rapidamente cresceu entre 1985 até meados da década de 1990, quando surgem os primeiros concorrentes sérios. Em 1993, Actel dominava cerca de 18 por cento do mercado.

Os anos 90 foram um período explosivo para os FPGAs, tanto em termos de sofisticação como de volume de produção. No início de 1990, os FPGAs eram utilizados principalmente em telecomunicações e redes. Pelo final da década, os FPGAs encontraram o caminho para a várias indústrias e aplicações dedicadas ao consumidor.

Os FPGAs têm um vislumbre da fama em 1997, quando Adrian Thompson, um cientista que trabalha na Universidade de Sussex, Inglaterra, fundiu a tecnologia algoritmo genético e FPGAs para criar um dispositivo de reconhecimento de som. O algoritmo de Thomson configurava uma matriz de 10 x 10 células num chip FPGA da Xilinx para distinguir entre dois

tons, utilizando recursos analógicos do chip digital. A aplicação de algoritmos genéticos para a configuração de dispositivos como o FPGA é agora referido como hardware Evolvable.» [1]

Através dos dados fornecidos em [1], é possível traçar o gráfico que mostra a evolução da produção de FPGAs ao longo dos últimos 25 anos:

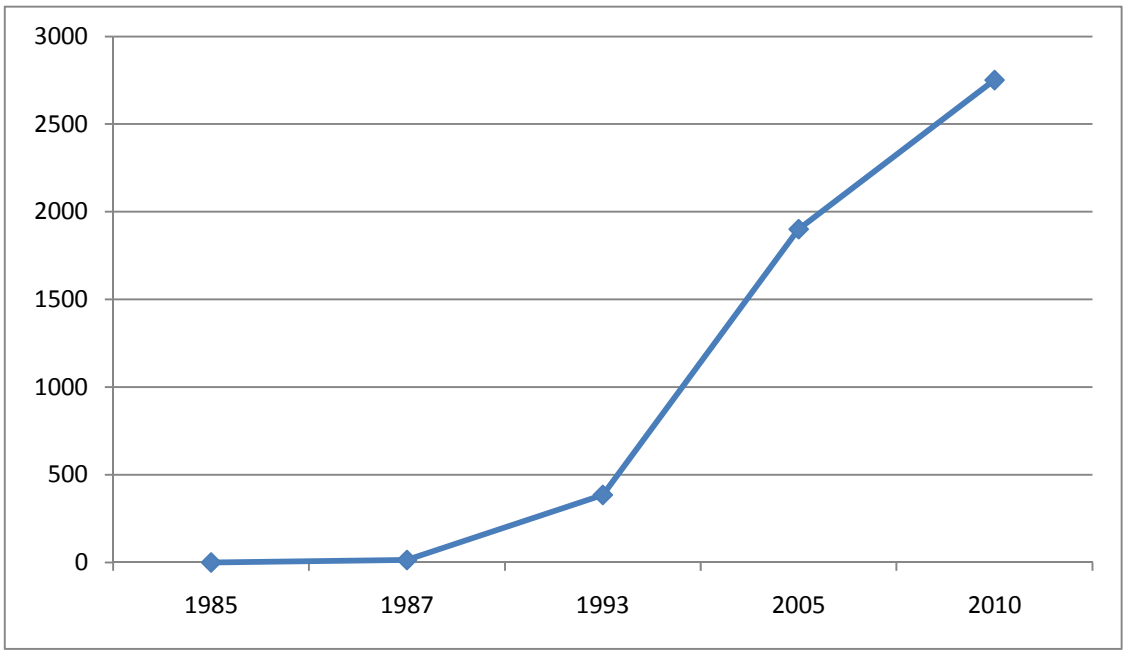


Figura 2.2 – Evolução da produção de FPGAs em milhões ao longo dos anos

2.1.1.2 - Aplicações

Actualmente, existem inúmeras aplicações de FPGAs, entre elas o processamento de sinal digital, rádio definido por software, sistemas aeroespacial e de defesa, protótipos ASIC, imagens médicas, visão computacional, reconhecimento de voz, criptografia, bioinformática, emulação de hardware de computador, radioastronomia, detecção de metais e um vasto leque de aplicações noutras áreas.

Os FPGAs podem ser encontrados mais facilmente em aplicações que necessitam de executar várias funções em paralelo uma vez que esta arquitectura disponibiliza um paralelismo massivo.

Existem duas grandes limitações à utilização de FPGAs: a complexidade do projecto quando comparado com software convencional e o tempo de compilação e/ou debug das ferramentas de programação actuais, que poderá demorar até várias horas após uma ligeira alteração do código fonte.

2.1.1.3 - Arquitectura

A arquitectura de um FPGA pode variar consoante o fabricante, mas na generalidade, são todas uma variação da arquitectura apresentada na imagem seguinte. Esta arquitectura consiste em blocos de entrada/saída configuráveis, blocos lógicos configuráveis e matriz de interligação configurável.

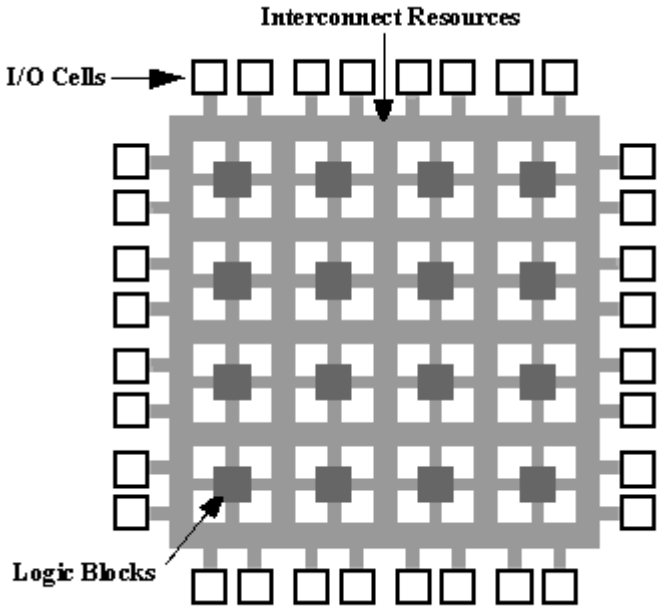


Figura 2.3 – Arquitectura interna de um FPGA [2]

Os blocos de entrada e saída são os circuitos responsáveis por enviar o sinal para o chip e por devolver a saída aos terminais, por outras palavras consiste num buffer de entrada e noutro de saída.

Os blocos lógicos configuráveis contêm a lógica de um FPGA. Estes contêm matrizes de pesquisa de 4 ou 6 entradas (LUT), flip-flops, *clock* e multiplexadores.

A matriz de interligações configurável é a matriz responsável pela programação interna das interligações de modo a seguir a necessidade do programador.

2.1.1.4 - Projecto e Programação

De modo a configurar o FPGA, o utilizador fornece um desenho esquemático, ou um código em HDL (hardware description language). O HDL torna-se mais fácil de trabalhar para projectos maiores, pois é possível especificar todas as estruturas apenas numericamente ao invés de ter que desenhar cada parte à mão. Por outro lado, a entrada esquemática pode permitir a fácil visualização de um projecto, tornando-se fácil de compreensão para um projecto de pequenas dimensões.

As linguagens HDL mais utilizadas são VHDL, Verilog e ABEL.

De modo a tentar reduzir a complexidade e o tempo de desenvolvimento de protótipo, a National Instruments LabView FPGA oferece uma programação gráfica de alto nível.

2.1.2 - Microcontroladores

Um microcontrolador é um circuito integrado composto por um processador (CPU) relativamente simples combinado com funções de suporte tais como oscilador de cristal, temporizadores, temporizador de *watchdog*, porta série, portos de entrada e saída analógicos, etc. Os microcontroladores são desenhados para pequenas aplicações ou aplicações dedicadas. Assim, em contraste com os microprocessadores utilizados nos computadores pessoais, a simplicidade é valorizada. Existem microcontroladores que conseguem operar abaixo dos 4kHz possibilitando aplicações de baixo consumo (na ordem dos mili ou microwatts). Geralmente têm a capacidade de manter a funcionalidade enquanto espera por um evento, fazendo com que bem adaptados, podem ser usados em aplicações onde a longa duração da bateria é valorizada.

Outros microcontroladores podem ter papéis críticos onde a performance é necessária, em que o oscilador terá que funcionar a uma frequência mais elevada e o consumo será maior.

Figura 2.4 – Intel 8048: o primeiro microcontrolador

2.1.2.1 - História

«Em 1969, uma equipa de engenheiros da empresa japonesa BUSICOM chegou aos Estados Unidos com o pedido de alguns circuitos integrados para calculadoras ser feito utilizando os seus projectos. A proposta foi entregue à INTEL e Marcian Hoff foi o responsável pelo projecto. Como que ele foi quem teve a experiência de trabalhar com um computador (PC) PDP8, ocorreu-lhe sugerir uma solução substancialmente diferente em vez da construção sugerida. Esta solução pressupunha que a função do circuito integrado seria determinada por um programa nele armazenado. Isso significava que a configuração seria mais simples, mas que exigiria mais memória do que o projecto que foi proposto pelos engenheiros japoneses.

A Intel continuou com o desenvolvimento e em Abril de 1974 lançaram para o mercado o processador de 8 bits com um nome de 8080, que foi capaz de endereçar 64KB de memória, e que tinha 75 instruções, e os preços a começarem nos $360.» [3]

Contudo, os microprocessadores precisavam de componentes externas para implementar um sistema operacional, aumentando o custo final e impossibilitando aplicações computorizadas económicas.

O primeiro sistema de computador num só chip optimizado para aplicações de controlo foi o microcontrolador Intel 8048 lançado em 1975, com RAM e ROM no mesmo integrado. Este microcontrolador foi introduzido em cerca de mil milhões de teclados de computador e em outras numerosas aplicações.

Por esta altura, os microcontroladores tinham duas variantes: EEPROM e PROM.

A EEPROM era significativamente mais cara do que a PROM. Em 1993, a introdução da EEPROM permitiu aos microcontroladores apagarem de um modo rápido os dados da memória sem recurso a nenhum dispositivo caro como acontece no caso das EPROM, permitindo uma programação on-board. No mesmo ano, a Atmel apresentou o primeiro microcontrolador que utilizava uma memória flash.

Actualmente os microcontroladores são de baixo custo e prontos para serem utilizados por quem goste, com uma comunidade online vasta orientada para certos processadores.

2.1.2.2 - Aplicações

Os microcontroladores são encontrados em quase todos os aparelhos electrónicos denominados "inteligentes". Desde os microondas convencionais aos sistemas de travagem automóvel, estão à nossa volta a executar tarefas que fazem a nossa vida mais cómoda e segura. Os microcontroladores são essencialmente pequenos computadores. Ao contrário dos computadores desktop convencionais, os microcontroladores interagem com outras máquinas ao invés de interagir com pessoas. Uma aplicação possível será a medição da temperatura da máquina de café, ou o sistema de protecção de sobreaquecimento de uma torradeira, etc.

Um microcontrolador poderá também ser usado para gerir o número de lugares disponíveis num parque de estacionamento, mostrando luz verde ou vermelha nos lugares livres e ocupados respectivamente e indicando o caminho para lugares livres, tal como se vê nos centros comerciais actualmente.

2.1.2.3 - Arquitectura

Um microcontrolador é um circuito integrado único, que inclui no seu interior as três unidades funcionais de um computador: unidade central de processamento, memória e portos de entrada/saída.

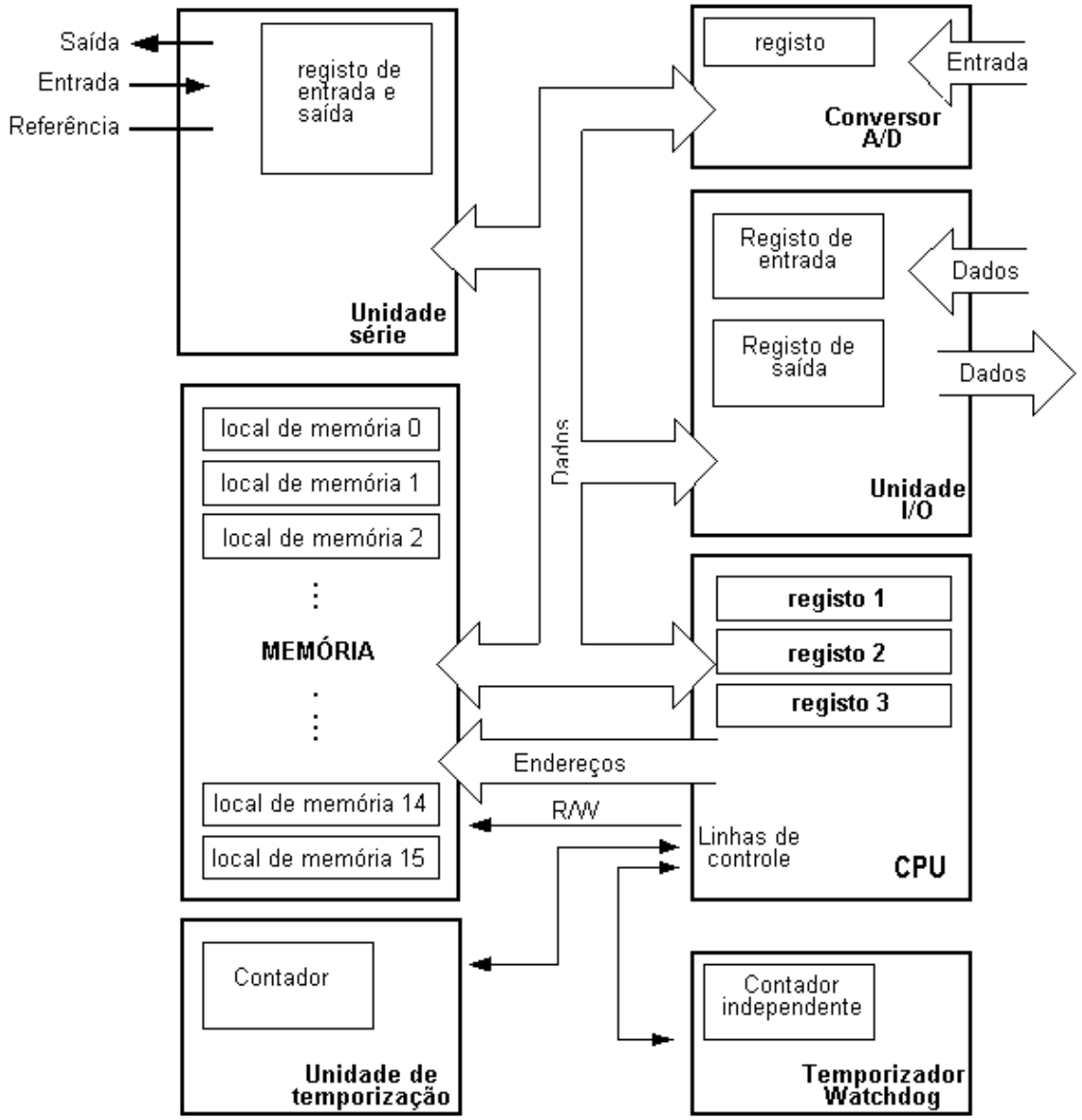


Figura 2.5 – Esquema de um microcontrolador [3]

Um microcontrolador típico contêm um gerador de *clock* integrado, uma pequena quantidade de memória RAM, que apesar de ser uma memória volátil, permite armazenar dados ou instruções de programa, assim como também contém uma pequena quantidade de memórias não voláteis que podem ser ROM/EPROM/EEPROM/Flash. A ROM é uma memória não volátil que apenas permite a leitura e responsável pelo armazenamento do firmware. As memórias EPROM, EEPROM e Flash servem para armazenamento de dados, em que a EPROM apenas é apagada através de exposição prolongada a raios UV. A EEPROM e a flash são electricamente apagáveis, sendo que a memória flash é uma memória de maior velocidade de acesso. Os microcontroladores geralmente também dispõem de uma grande variedade de dispositivos de entrada/saída tais como conversores analógico/digital, temporizadores, e protocolos de comunicação, tais como UARTs e barramentos de série especializados como I2C

e CAN. Frequentemente estes dispositivos integrados podem ser controlados por instruções de processadores especializados.

2.1.2.4 - Ambientes de Programação

Originalmente, os microcontroladores eram programados apenas em Assembly. Actualmente também é possível programar utilizando linguagens de alto nível. Estas linguagens são projectadas especialmente para o efeito, ou são versões de linguagem de propósito geral, tais como a linguagem de programação C. Os compiladores para linguagens de propósito geral normalmente têm algumas restrições, bem como melhorias para facilitar a interacção com as características do microcontrolador.

Alguns microcontroladores têm ambientes para ajudar a desenvolver certos tipos de aplicação. Actualmente, os fabricantes disponibilizam frequentemente ferramentas de desenvolvimento gratuitas para tornar mais fácil a adopção do seu hardware.

Existem também simuladores disponíveis para alguns microcontroladores, tal como acontece com o ambiente da Microchip MPLAB, que permite que um programador analise o comportamento do microcontrolador e do seu programa caso estivesse a usar equipamento real. O simulador mostra o estado do processador, assim como o estado das saídas, bem como permite gerar sinais de entrada. Enquanto por um lado a maioria dos simuladores estão limitados à simulação com muitos outros equipamentos hardware, estes podem simular condições que seriam difíceis de implementar num sistema físico e podem ser a maneira mais rápida de depurar e analisar problemas.

Os microcontroladores recentes são frequentemente integrados com um modo de depuração “on chip” que permite a depuração na própria placa.

2.2 - Tecnologia de Comunicação sem fios

Actualmente, as soluções de redes sem fios estão a substituir as existentes redes de cabos convencionais e estão a ser criadas novas aplicações para redes sem fios.

Os avanços recentes das tecnologias de redes sem fios possibilitaram o aparecimento de várias alternativas e padrões de implementação, mas até recentemente, a grande maioria tinha como objectivo principal fornecer um conjunto de protocolos que garantissem a qualidade para a transmissão de voz ou de dados com altas taxas de transferência, o que tornava os equipamentos bastante caros e pouco atraentes para aplicações mais simples.

Apesar da evolução tecnológica, ainda não existem muitos padrões de redes sem fio para aplicações em redes locais utilizando sensores e/ou outros dispositivos de controlo. O que existe são basicamente sistemas proprietários, desenvolvidos para atender redes específicas,

tais como as redes de automação industrial, onde aplicações com sensores e dispositivos de controlo não necessitam de uma largura de banda elevada, mas necessitam de uma latência e consumo baixo para preservar a vida útil das baterias.

As recomendações do IEEE, particularmente as recomendações da série IEEE 802.11, são os exemplos mais conhecidos para os padrões de redes sem fio e que nos permitem considerar a existência de quatro grandes grupos: [6]

- **WPAN (Wireless Personal Area Network) –** Onde se encontram as tecnologias sem fio de pequeno alcance (entre 10 e 100 metros). Entre as tecnologias abordadas por este padrão estão incluídas as tecnologias Zigbee e Bluetooth;
- **WLAN (Wireless Local Area Network) –** Onde se encontram as tecnologias sem fio destinadas à interligação de redes locais com alcance entre 100 e 300 metros. Trata-se de padrão implementado como extensão ou alternativa para as redes de cabos convencionais. A tecnologia mais conhecida deste padrão é a WiFi, utilizada diariamente por milhões de utilizadores;
- **WMAN (Wireless Metropolitan Area Network) -** Neste grupo encontram-se as tecnologias que tratam dos acessos de banda larga para redes em áreas metropolitanas, com alcance em cerca de 6km;
- **WWAN (Wireless Wide Area Network) –** Neste grupo encontram-se as tecnologias viradas para redes de longa distância em telecomunicações, atendendo aos serviços de voz e alguns serviços de dados.

De acordo com os requisitos iniciais do projecto, e com base na lista dos padrões de rede sem fios acima mencionados, o padrão WPAN é o indicado, tendo em conta que é necessário um alcance relativamente reduzido (o sistema tem que ser capaz de cobrir eficazmente um salão de jogos) e uma baixa taxa de transmissão. Das tecnologias disponíveis para este padrão, as mais emergentes são o Bluetooth e o ZigBee, sendo que a primeira é mais indicada para taxas de transmissão mais elevadas e a segunda mais indicada para redes de nós. Consequentemente, em seguida é analisada a tecnologia ZigBee.

2.2.1 - ZigBee

O Zigbee é uma tecnologia de baixo custo e de baixo consumo que permite uma ligação múltipla sem fios que é altamente usada em aplicações de controlo e monitorização. O facto de ser de baixo consumo permite que utilizando pequenas baterias, o equipamento seja de longa duração e como permite múltiplas ligações e roteamento de dados, fornece alta fiabilidade e um longo alcance.

O Zigbee opera nas bandas 868MHz na Europa, 915Mhz nos EUA e Austrália e 2,4GHz em todo o mundo. Como este pode ser activado (ir desde o modo sleep ao activo) em menos de 15mseg, a latência pode ser muito baixa e os dispositivos serem rápidos a responder. O protocolo Zigbee actualmente foca-se num propósito geral, de baixo custo, cuja auto-

organização da rede de malha pode ser usada para controlo de habitações (iluminação inteligente, controlo de temperatura avançada, segurança, etc.), sensores (de água, energia, monitorização de energia, detectores de fumo e fogo, aparelhos de acesso, etc.), etc.

O padrão ZigBee foi desenvolvido para se tornar numa alternativa de comunicação em redes que não necessitem de soluções mais complexas para o seu controlo, baixando assim os custos de aquisição, instalação de equipamentos, manutenção e mão-de-obra. Trata-se de uma tecnologia relativamente simples, que utiliza um protocolo de pacotes de dados com características específicas, sendo projectado para oferecer flexibilidade quanto aos tipos de dispositivos que pode controlar.

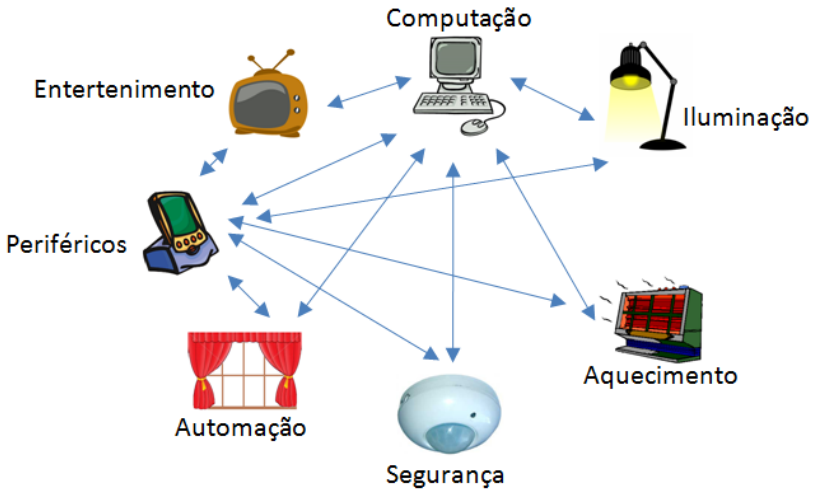


Figura 2.6 – Aplicações do ZigBee

2.2.1.1 - História

As redes tipo Zigbee começaram a ser concebidas em 1998 quando se chegou à conclusão que quer a WiFi quer o Bluetooth eram tecnologias inadequadas a muitas aplicações, em particular aplicações em que era necessária a auto-organização da rede.

Em Maio de 2003, a norma IEEE 802.15.4 foi concluída. No verão do mesmo ano, A Philips Semiconductors, que era um dos maiores apoiantes desta tecnologia, pararam o investimento, contudo a empresa Philips Lightning continuou a participação da Philips e esta manteve-se no quadro de direcção da Aliança Zigbee.

A Aliança Zigbee anunciou em Outubro de 2004 que o número de membros tinha duplicado desde o ano anterior e que tinha crescido para mais de 100 empresas membros espalhados por 22 países. Em Abril de 2005 a parceria já tinha ultrapassado as 150 empresas em Dezembro do mesmo ano já tinha ultrapassado as 200.

As especificações Zigbee foram ratificadas em 14 de Dezembro de 2004. A 13 de Junho de 2005, a Aliança Zigbee anuncia a disponibilização 1.0 denominada Especificação Zigbee 2004. Dois anos depois, anuncia a disponibilização completa e imediata aos seus membros a versão Zigbee melhorada, denominada Especificação Zigbee 2006.

Durante o último quarto de 2007 essa especificação foi finalizada e ficou a ser conhecida por Zigbee Pro. [5]

2.2.1.2 - Funcionamento

Os dispositivos baseados na tecnologia ZigBee operam na faixa ISM que não requer licença para funcionamento, incluindo as faixas de 2,4GHz (Global), 915Mhz (América) e 868Mhz (Europa) e com taxas de transferência de dados de 250kbps em 2,4GHz, 40kbps em 915Mhz e 20kbps em 868Mhz. [7]

Esta tecnologia oferece actualmente interfaces com velocidades de ligação compreendidas entre 10Kbps e 115Kbps e com um alcance de transmissão entre 10m e 100m, dependendo directamente da potência dos equipamentos e de características ambientais (obstáculos físicos, interferência electromagnética, etc.).

Quanto ao problema de alimentação dos dispositivos, os módulos de controlo dotados com esta nova tecnologia podem ser alimentados por baterias comuns, sendo que a sua vida útil está relacionada directamente com a capacidade da bateria e a aplicação a que se destina. Nesse aspecto, o protocolo ZigBee foi projectado para suportar aplicações com o mínimo de consumo (com pilhas convencionais AA, um dispositivo pode funcionar até 6 meses).

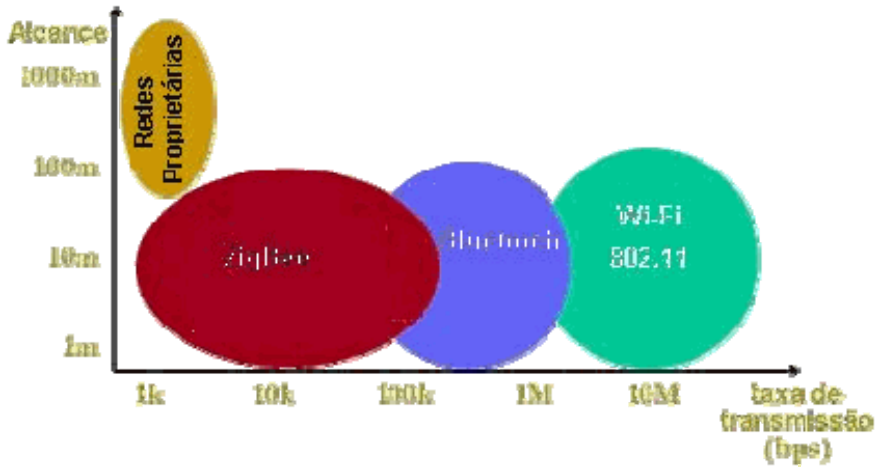


Figura 2.7 – Comparação do ZigBee com outras tecnologias [6]

2.2.1.3 - Tipos de Dispositivos e Topologias de Rede

Tabela 2.1 — Tipos de dispositivos com base nas definições IEEE [7]

Tipo de dispositivo	Serviços disponibilizados
Full Function Device (FFD)	Quase todos ou todos
Reduced Function Device (RFD)	Limitado

Tabela 2.2 — Tipos de dispositivos de protocolo [7]

Dispositivo de protocolo	Tipo de Dispositivo	Funções Típicas
Coordenador	FFD	Um por rede. Cria a rede, aloca os endereços de rede e permite que outros dispositivos se liguem à rede.
Router	FFD	Opcional. Extende o alcance da rede. Permite que mais nós se juntem à rede. Pode também monitorizar e/ou efectuar funções de controlo.
Dispositivo Final	FFD ou RFD	Monitoriza e/ou efectua funções de controlo.

Figura 2.8 – Legenda para as topologias em Zigbee

Podemos identificar três tipos topologias de rede: topologia em estrela, topologia do tipo em árvore e topologia emalhada.

A topologia em estrela consiste num nó coordenador ZigBee e num ou mais dispositivos finais. Neste tipo de ligação, todos os dispositivos finais comunicam apenas com o coordenador. Se algum dispositivo final precisar de enviar informação para um outro, este envia para o coordenador e este último envia para o destinatário. [7]

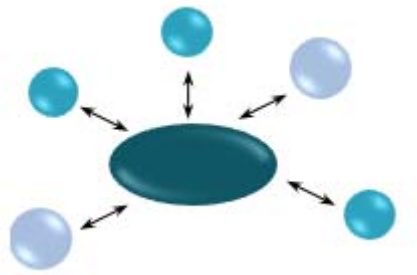


Figura 2.9 – Topologia de rede em estrela

Outro tipo de topologia de rede é a topologia em árvore. Nesta topologia, os dispositivos finais podem-se ligar à rede através do coordenador ZigBee ou dos routers ZigBee. Os routers têm duas funções: incrementar o número de nós permitidos pela rede e estender fisicamente o alcance da rede. Com a adição de um router, o dispositivo final não precisa de estar no alcance directo do coordenador. Todas as mensagens nesta topologia são roteadas ao longo de toda a árvore. [7]

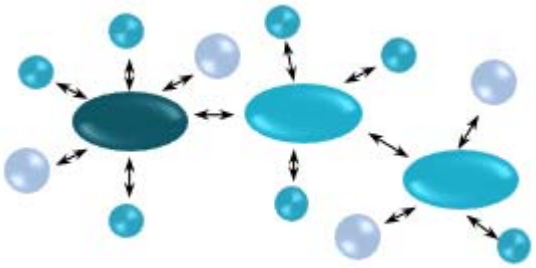


Figura 2.10 – Topologia de rede do tipo árvore

Existe também outro tipo de topologia que é denominada por rede emalhada. Este tipo de topologia difere da topologia do tipo árvore no aspecto de todos os nós podem comunicar com qualquer outro nó, isto é, todos os nós têm o mesmo nível de acesso aos meios de comunicação. [7]

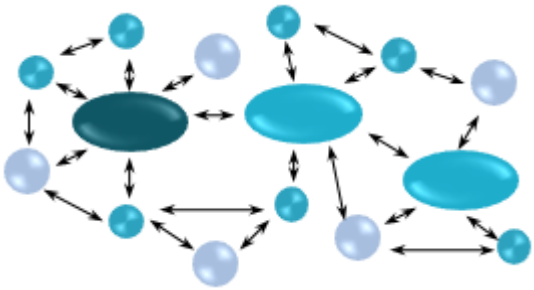


Figura 2.11 – Topologia de rede emalhada

2.2.1.4 - Características

O padrão ZigBee (IEEE 802.15.4) foi projectado de modo a que o consumo de potência fosse baixo e que fosse de implementação simples, com interfaces de baixo custo. Permite o funcionamento em dois estados: "active" para transmissão e/ou recepção e "sleep", quando não está transmitindo. Existe uma simplicidade de configuração e uma protecção contra a redundância de dispositivos (operação segura). A densidade de nós por rede é elevada uma vez que as camadas PHY e MAC permitem que as redes funcionem com grande número de dispositivos activos. Este atributo é crítico para aplicações com sensores e redes de controlo. Este protocolo acaba por ser simples, o que permite a transferência confiável de dados com os níveis de segurança apropriados.

2.2.1.5 - Camada de Protocolos

A publicação do padrão IEEE 802.15.4, definiu os interfaces com baixas taxas de transmissão (menores que 250Kbps) e estabeleceu uma estrutura de rede que incorpora os conceitos de redes *ad-hoc*, características de ligação em malha e em *multi-hop* (múltiplos saltos). Adicionalmente, novos algoritmos de segurança e perfis de aplicação foram definidos de modo a garantir a segurança e a perfeita interacção entre os diversos equipamentos.

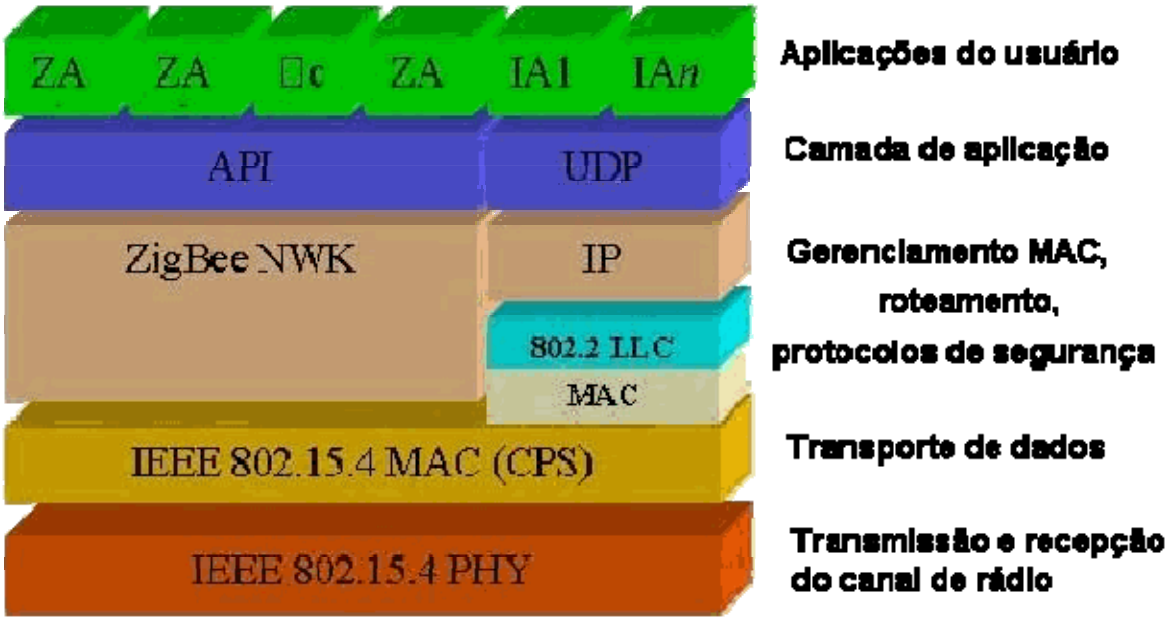


Figura 2.12 – Camadas de protocolos ZigBee [6]

A camada física (PHY) foi projectada para acomodar as necessidades de interfaces de baixo custo, permitindo níveis elevados de integração. O uso da técnica de transmissão de Sequência Directa (DSS) permite que os equipamentos sejam muito simples, possibilitando implementações mais baratas.

A camada do Media Access Control (MAC) foi projectada para permitir múltiplas topologias de baixa complexidade, onde o gerenciamento de energia, por exemplo, não requer modos de operação complexos. O MAC também permite que um dispositivo RFD opere na rede sem a necessidade de grandes quantidades de memória disponíveis, podendo controlar também um grande número de dispositivos sem a necessidade de colocá-los "em espera", como ocorre em algumas tecnologias sem fio.

A camada de rede foi projectada para possibilitar o crescimento da rede sem a necessidade de equipamentos de transmissão de potência mais elevada. A camada de rede também pode operar grandes quantidades de nós de rede com latências relativamente baixas.

A camada NWK utiliza um algoritmo que permite implementações da pilha de protocolos visando balancear os custos das unidades em aplicações específicas, o consumo das baterias, com o objectivo de produzir soluções com o perfil específico de custo-desempenho para a aplicação. [8]

2.2.1.6 - Segurança

O padrão ZigBee utiliza chaves de 128 bits para implementar os seus mecanismos de segurança.

A camada MAC utiliza o padrão AES como seu algoritmo de criptografia, descrevendo uma variedade de rotinas de segurança. Estas rotinas têm como objectivo prover a confidencialidade, a integridade e a autenticidade das frames da camada MAC. A camada MAC faz o processamento de segurança, mas são as camadas superiores que controlam o processo, ajustando as chaves de criptografia e determinando os níveis de segurança que deverão ser usados. Quando a camada MAC transmite (ou recebe) uma frame, verifica o destino (ou a fonte da frame), recupera a chave associada com esse destino (fonte), e usa então esta chave para processar a frame de acordo com a rotina de segurança designada para a chave que está sendo usada. Cada chave é associada a uma única rotina de segurança e o cabeçalho da frame MAC possui um bit que especifica se a segurança para a frame está habilitada ou não. [6]

2.2.1.7 - Tipos de Tráfego

O padrão suporta diferentes tipos de tráfego de dados que exigem atributos diferentes da camada MAC. O MAC IEEE 802.15.4 é flexível o bastante para assegurar o transporte de cada um dos tipos de tráfego como;

- Dados periódicos, provenientes de sensores;
- Dados intermitentes, provenientes de interruptores e chaves;
- Dados provenientes de dispositivos repetitivos de baixa latência como, por exemplo, um rato. [6]

Para além da tecnologia ZigBee, existem várias tecnologias baseadas no mesmo protocolo mas com alterações a nível das camadas, normalmente simplificando, de modo a serem tecnologias menos complexas. A MiWi é uma dessas tecnologias, que sendo baseada na tecnologia ZigBee, têm algumas limitações, que poderão ou não ter implicações no projecto em desenvolvimento.

2.2.2 - MiWi

A implementação de aplicações de rede sem fios está-se a tornar comum. Desde os dispositivos comuns até às aplicações industriais, existe uma crescente expectativa de que os dispositivos terão embutido a capacidade de comunicar uns com os outros através da

tecnologia sem fio. O desafio é escolher o protocolo de rede sem fio mais adequado e aplicá-lo de um modo que seja efectivo relativamente ao custo. O MiWi Wireless Networking Protocol é um protocolo simples projectado para baixas taxas de transmissão de dados, curtas distâncias, e redes de baixo custo. Baseada no protocolo IEEE 802.15.4 para redes de área pessoal (WPANs), o protocolo de MiWi fornece-nos uma alternativa fácil de usar para redes sem fio.

Em particular, o MiWi tem como alvo pequenas aplicações com redes relativamente pequenas com baixas transmissão de dados, utilizando o transceptor MRF24J40 2.4GHz da Microchip para redes compatíveis.

Uma rede utilizando este protocolo é capaz de ter um máximo de 1024 nós. Cada coordenador é capaz de suportar um máximo de 128 nós adjacentes, com um máximo de 8 coordenadores por rede.

2.2.2.1 - Funcionamento

O protocolo MiWi é baseado nas camadas MAC e PHY da especificação IEEE 802.15.4 e foi ajustada para um desenvolvimento simples de uma rede de 2.4GHz. O protocolo fornece comandos para procurar, ligar ou criar uma rede, e também de descobrir nós na rede ou até mapeá-los. Este protocolo não especifica que rede se liga nem quantas vezes um dispositivo deve comunicar.

A especificação IEEE 802.15.4 disponibiliza três bandas de frequências, sendo que cada uma delas oferece um determinado número de canais e tem uma taxa de transmissão máxima.

Tabela 2.3 — Especificações das bandas de frequência [9]

Banda de Frequência	Canais Disponíveis (Nº Canal)	Transmissão máxima de dados (kbps)
868 MHz	1 (0)	20
915 MHz	10 (1-10)	40
2.4 GHz	16 (11 – 26)	250

O tamanho máximo de um MAC packet na especificação IEEE 802.15.4 é de 127 bytes, incluindo um valor 16 bits de CRD. Este valor verifica a integridade da frame.

O IEEE 802.15.4 opcionalmente utiliza um mecanismo de transferência de dados no MAC com conhecimento. Este método uma flag especial ACK no cabeçalho do packet. Quando esta flag está ligada, o conhecimento para o transmissor é necessário; isto assegura que a frame é, de facto, entregue. Se a frame é transmitida com uma flag ACK e o conhecimento não for recebido num determinado período, o transmissor irá tentar novamente a transmissão um determinado numero de vezes antes de declarar erro.

É importante notar que a recepção do conhecimento apenas indica que a trama foi correctamente recebida. Ela não indica, contudo, que a trama foi bem processada.

2.2.2.2 - Tipos de Dispositivos e Topologias de Rede

O protocolo MiWi define três tipos de dispositivos, baseado com as suas funções na rede: Coordenador PAN, Coordenador e Dispositivo final. Os tipos de dispositivos e as suas relações podem ser mostrados na tabela seguinte.

Tabela 2.4 — Tipos de dispositivos com base nas definições IEEE [9]

Tipo de dispositivo	Serviços disponibilizados
Full Function Device (FFD)	Quase todos ou todos
Reduced Function Device (RFD)	Limitado

Tabela 2.5 — Tipos de dispositivos de protocolo [9]

Tipo de Dispositivo	Tipo de dispositivo IEEE	Funções Típicas
Coordenador PAN	FFD	Um por rede. Cria a rede, aloca os endereços de rede e guarda registo dos endereços numa tabela
Coordenador	FFD	Opcional. Extende o alcance físico da rede. Permite que mais nós se juntem à rede. Pode também executar funções de monitorização e/ou controlo
Dispositivo Final	FFD ou RFD	Executa funções de monitorização e/ou controlo

Dos três tipos de dispositivos definidos no protocolo MiWi, o mais importante é o coordenador PAN. Este dispositivo cria a rede, selecciona o canal e o PAN ID. Todos os outros dispositivos que se ligam têm que obedecer às instruções do coordenador PAN.

Podemos identificar três tipos topologias de rede: topologia em estrela, topologia do tipo árvore e topologia emalhada.

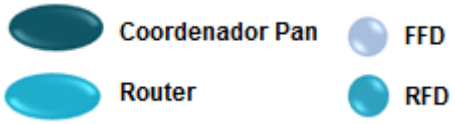


Figura 2.13 – Legenda para as topologias MiWi

Uma topologia em estrela consiste num nó coordenador PAN e num ou mais dispositivos finais. Neste tipo de topologia, todos os dispositivos finais comunicam apenas com o coordenador PAN. Caso um dispositivo final tenha que transferir informação para outro dispositivo final, este terá que enviar primeiro para o coordenador PAN e este é que reenvia a informação para o dispositivo final destinatário. [9]

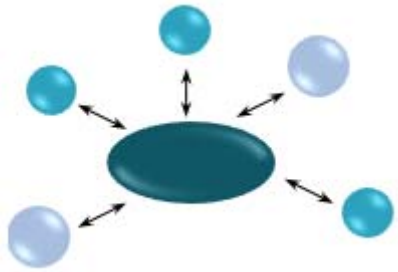


Figura 2.14 – Topologia em Estrela

Numa topologia do tipo árvore, existe apenas um coordenador PAN, contudo outros coordenadores podem-se ligar à rede. Deste modo, uma estrutura do tipo árvore é criada, onde o coordenador PAN é a raiz da árvore, os coordenadores são os ramos e os dispositivos finais são as folhas. Todas as mensagens enviadas nesta topologia devem ser roteadas conforme a estrutura. Como as mensagens normalmente são roteadas através de mais que um nó para chegar ao destino, esta configuração é por vezes denominada por redes multi-salto. [9]

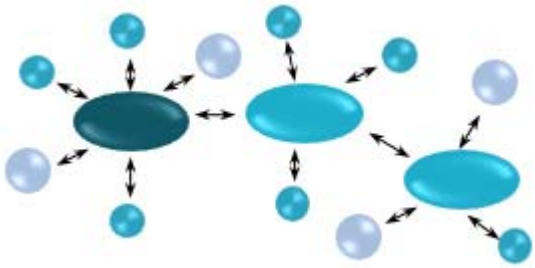


Figura 2.15 – Topologia do tipo árvore

Uma topologia de rede emalhada é similar a uma rede do tipo árvore, excepto que os FFDs podem rotear mensagens directamente para outros FFDs em vez de seguir pela estrutura. As mensagens para RFDs continuam a ser enviadas pelo nó antecessor. As vantagens desta topologia são que a latência pode ser reduzida e a fiabilidade pode ser aumentada. Tal como acontece com as redes do tipo árvore, as redes emalhadas são multi-salto. [9]

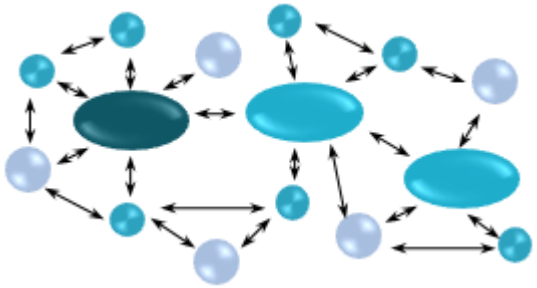


Figura 2.16 – Topologia em rede emalhada

2.2.2.3 - Comparação com ZigBee

O protocolo MiWi é bastante simplificado quando comparando com o ZigBee, o que implica um espaço de memória inferior ao necessário para utilizar a tecnologia ZigBee.

Tabela 2.6 — Diferenças entre MiWi e ZigBee

	MiWi	**Zigbee**
Tamanho do Código [10]:	Coordenador: 7,1 kbytes Router: 7,1 kbytes Dispositivo Final: 5,2 Kbytes	Coordenador: 29 kbytes Router: 30,7 kbytes Dispositivo Final: 23,5 kbytes
Especificação:	Disponível online como nota de aplicação	Especificação aberta. Standardizado
Rede:	1024 nós máximos 4 saltos máximos	65 536 nós máximos infinitos saltos
Requisitos	Utilizar o MRF24J40 da Microchip	Existe hardware de vários fabricantes
Orientado para:	Pequenas redes	Redes de grandes dimensões

Para além da MiWi existem mais tecnologias baseadas no ZigBee. O protocolo MiWi P2P é uma simplificação do protocolo MiWi e consiste na remoção do elemento intermédio *router*. Esta tecnologia, apesar de ser limitativa, tem as suas vantagens e será analisada de seguida:

2.2.3 - MiWi P2P

Actualmente a procura de soluções sem fio está a aumentar exponencialmente. As vantagens são custos reduzidos e uma fácil implementação. A comunicação sem fio não requer cablagem ou qualquer outro hardware associado à instalação. Pode também ser utilizada em situações onde efectuar uma cablagem seria muito difícil ou até mesmo impossível.

Desde que o IEEE lançou, em 2003, as especificações WPAN (Wireless Personal Area Network – IEEE 802.15.4) que se tornou na indústria standard para redes WPAN de baixas transmissões.

O protocolo Microchip MiWi P2P é uma variação do IEEE 802.15.4 que utiliza O transceiver da Microchip MRF24j40 e qualquer outro microcontrolador Microchip com comunicação SPI.

Este protocolo fornece uma comunicação directa e fiável através de uma interface de fácil programação. Existe uma pilha de recursos que pode ser compilada de modo a ir ao encontro das especificações do consumidor, minimizando o tamanho de código.

A camada MAC é modificada por este protocolo, na medida em que são adicionados comandos que simplificam o processo de comunicação. Isso simplifica desligar um nó e o salto de canais fornecendo comandos MAC adicionais. Contudo, decisões específicas tais como quando executar a detecção de energia ou quando saltar de canal não são definidas no protocolo, sendo deixadas ao cuidado do programador.

2.2.3.1 - Funcionamento

O protocolo MiWi P2P fornece 16 canais na gama de 2.4GHz (utilizando o transceiver MRF24J40), opera com qualquer microcontrolador da Microchip com comunicação SPI, suporta os compiladores C18, C30 e C32, suporta o modo sleep, procura o canal com menos ruído para operar, fornece uma pesquisa activa para detectar ligações existentes, suporta todos os modos definidos pela especificação IEEE 802.15.4 e permite uma boa agilidade de frequência (saltos entre canais).

Este protocolo é uma variação do IEEE 802.15.4 e suporta as topologias P2P e em estrela. Não permite nenhum sistema de roteamento, por consequente a cobertura da comunicação está limitada pelo alcance das unidades rádio. O tempo de intervalo garantido e as redes do tipo *beacon* não são suportadas, por isso ambos os lados da comunicação não podem entrar no modo sleep simultaneamente.

A pilha MiWi P2P usa apenas uma porção das especificações IEEE 802.15.4 referentes ao PHY e MAC. A especificação define três camadas PHY, operando num espectro de 868 MHz, 915 MHz e 2,4GHz. Esse espectro contém 16 canais disponíveis e um tamanho máximo do packet de 127 bytes, incluindo dois bytes relativos ao CRC (cyclic redundancy check).

O total da largura de banda para o IEEE 802.15.4 relativamente aos 2.4GHz é teoricamente de 250kbps. Na realidade, para comunicações fiáveis, a largura ronda os 20-30kbps. [11]

2.2.3.2 - Tipos de Dispositivos e Topologias de Rede

O protocolo MiWi P2P cataloga os dispositivos com base nas duas definições IEEE e no seu papel nas comunicações.

Tabela 2.7 — Tipos de dispositivos com base nas definições IEEE [11]

Tipo de dispositivo	Serviços disponibilizados
Full Function Device (FFD)	Quase todos ou todos
Reduced Function Device (RFD)	Limitado

Tabela 2.8 — Tipos de dispositivos de protocolo [11]

Tipo de Dispositivo	Tipo de dispositivo IEEE	Funções Típicas
Coordenador PAN	FFD	O dispositivo inicia primeiro e espera por uma ligação
Dispositivo Final	FFD ou RFD	O dispositivo inicia após o coordenador PAN ter estabelecido uma ligação

Figura 2.17 – Legenda para as topologias MiWi P2P

Uma topologia em estrela típica é mostrada na figura seguinte. Esta topologia tem um coordenador PAN (Personal Area Network) que inicia as comunicações e aceita as ligações de outros dispositivos. Os dispositivos finais apenas se podem ligar ao coordenador PAN.

Quanto à funcionalidade, o coordenador PAN é um FFD enquanto que os dispositivos finais podem ser FFD ou RFD com a sua unidade rádio desligada enquanto está em modo sleep. Independentemente do tipo funcional que é, apenas podem comunicar com o coordenador PAN. [11]

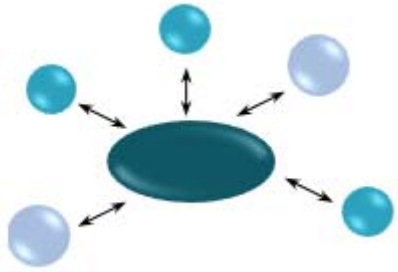


Figura 2.18 – Topologia do tipo estrela

Uma topologia P2P típica é mostrada na figura seguinte. Da perspectiva dos papéis que cada dispositivo executa, esta topologia tem um coordenador PAN que inicia as comunicações e tem dispositivos finais. Quando se ligam à rede, os dispositivos finais não necessitam se estabelecer uma ligação directa com o coordenador PAN. Da perspectiva funcional, o coordenador PAN é um FFD enquanto os dispositivos finais podem ser FFDs ou RFDs, podendo os FFD ter múltiplas ligações. Cada RFD pode-se ligar apenas a um FFD e não se pode ligar a nenhum RFD. [11]

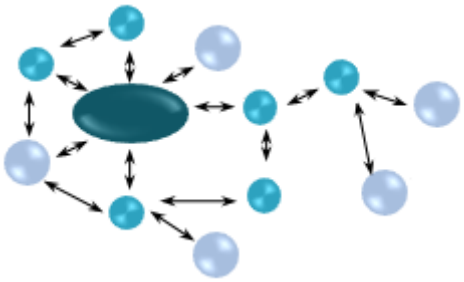


Figura 2.19 – Topologia Peer-To-Peer

2.2.3.3 - Comparação com MiWi

O protocolo MiWi P2P difere do protocolo MiWi no aspecto de não permitir router. Tal limitação reduz significativamente o alcance da rede e reduz o espaço de memória necessário quando comparado com a tecnologia MiWi.

Tabela 2.9 — Diferenças entre MiWi P2P e MiWi

	MiWi P2P	MiWi
Tamanho do Código [10]:	Coordenador PAN: 3,2 kbytes Dispositivo Final: 3,2 Kbytes	Coordenador: 7,1 kbytes Dispositivo Final: 5,2 Kbytes
Topologias:	Estrela e P2P	Estrela, do tipo árvore e emalhada
Rede:	1024 nós máximos em P2P ou 128 nós em árvore 4 saltos máximos	1024 nós máximos infinitos saltos

Neste capítulo foram apresentadas duas tecnologias de hardware, nomeadamente microcontroladores e FPGAs. Foi também apresentada a tecnologia Zigbee para comunicação sem fios, assim como duas alternativas simplistas (MiWi e MiWi P2P), mais apelativas para aplicações que requerem menor número de nós e menor flexibilidade. A partir da análise das tecnologias disponíveis para a implementação do sistema, pode-se avançar para a análise comparativa das mesmas de modo a poder concluir quais as tecnologias mais adequadas para o desenvolvimento do projecto. Para além da análise comparativa, o próximo capítulo irá descrever o desenvolvimento do projecto.

Capítulo 3

Projecto do sistema

Do capítulo anterior, resulta uma clara necessidade de um estudo comparativo das tecnologias abordadas, quer relativamente a hardware, quer relativamente às tecnologias de rede sem fios que permita concluir sobre a tecnologia de suporte ao desenvolvimernto do sistema. A análise é suportada com base nos requisitos estabelecidos para o sistema, baseando a decisão da escolha nos parâmetros de simplicidade de programação e futura produção.

Este capítulo está dividido em duas partes bem distintas: arquitectura de sistema, com estudo das características importantes para o sistema e comparação directa entre tecnologias e projecto e desenvolvimento do sistema, em que todos os passos do projecto são descritos.

Na primeira parte, organizada em quatro subcapítulos discute-se a escolha do *hardware* e *software a utilizar*, assim como se analisa detalhadamente o microcontrolador e unidade de rádio escolhida.

A segunda parte deste capítulo refere-se ao projecto e desenvolvimento. Este capítulo está dividido em dois subcapítulos: produção do circuito PCB, em que se apresenta o esquema, o desenho e a sua produção e algoritmos e estruturação do código. No primeiro subcapítulo, esquematizam-se as ligações, desenha-se a placa PCB, procede-se à produção do circuito impresso e respectiva montagem. No último subcapítulo, é feito um resumo dos algoritmos utilizados na placa de interface, assim como na placa do contador de tempo de bilhar e é apresentada a aplicação em Visual Basic que mostra o estado da rede de contadores de tempo de bilhar através da comunicação existente entre a placa de interface e o computador via porta série.

3.1 - Arquitectura de Sistema

Assumido que o sistema tem uma arquitectura distribuída, torna-se necessário fazer uma selecção precisa e apropriada do hardware e software a ser utilizado. Neste subcapítulo, foi efectuada uma análise comparativa entre o hardware e software utilizado, de modo a permitir concluir sobre a tecnologia a utilizar. Após a decisão, é também estudado o microcontrolador utilizado, assim como o módulo de unidade rádio e sua interligação.

3.1.1 - Hardware

Uma vez apresentadas as tecnologias de hardware disponíveis, é necessário proceder à selecção da tecnologia a utilizar. Para tal, foi criada uma tabela que comparasse as vantagens e desvantagens da utilização de cada tecnologia na arquitectura a ser desenvolvida.

Tabela 3.1 — Comparação directa entre microcontroladores e FPGAs

Microcontroladores		**FPGAs**	
Vantagens	Desvantagens	Vantagens	Desvantagens
Fácil Programação	Pagamento por funcionalidades não utilizadas	Performance Elevada	Programação e Debug complexas
Fiabilidade Garantida	Performance Reduzida	Fornecimento de FPGAs com a funcionalidade exacta	Utilização de energia constante
Modo Power Save			Alterações de design complicadas
Reduzido tempo de desenvolvimento			Elevado tempo de desenvolvimento
Alterações de design relativamente simples			
Baixo Custo			

Sabendo à partida que não é necessário um processamento elevado, pode-se observar na tabela anterior que o microcontrolador é a escolha mais apelativa. Decidida a tecnologia de hardware a utilizar, seria necessário escolher o microcontrolador ideal entre os milhares disponíveis no mercado e que permita a ligação de um módulo de unidade rádio compatível com a tecnologia de rede sem fios a ser utilizada.

A escolha do microcontrolador fica então pendente pela escolha do módulo de comunicação sem fios uma vez que se pretende uma inteligação directa. De modo a decidir sobre o módulo a utilizar, é necessária uma pesquisa sobre microcontroladores com módulo de comunicação incluídos e também pesquisa de módulos de comunicação sem fios para integração com um microcontrolador.

Partindo do pressuposto que a comunicação pretendida é baseada no protocolo IEEE 802.15.4, mais conhecida por WPAN, de modo a poder ser utilizado qualquer um dos protocolos ZigBee, MiWi e MiWi P2P, as alternativas passam a ser reduzidas, e são apresentadas nas duas tabelas seguintes:

Tabela 3.2.A — Comparação das alternativas sem fios

	TI CC2430	**TI CC2510F16**	**Microcontrolador + Microchip MRF24J40MA**	**Microcontrolador + Microchip MRF24J40MB**
Wireless :	IEEE802.15.4	IEEE802.15.4	IEEE802.15.4	IEEE802.15.4
Emissor :	+0 dBm	+0 dBm	+0 dBm	+20 dBm
Receptor :	-92 dBm	-103 dBm	-94 dBm	-102 dBm
Alcance :	??	??	120 m	1200 m
Vantagens :	All-In-One	All-In-One	Bom Suporte Web	Bom Suporte Web
Desvantagens :	Não é DIP	Não é DIP	Precisa de microcontrolador	Precisa de microcontrolador
Preço Unidade :	3,20 €	3,20 €	6,80 €	11,80 €
Preço Desenvolvimento :	375 €	450 €	188 €	188 €

Tabela 3.2.B — Comparação das alternativas sem fio

	Microcontrolador + Atmel ZibBit	Microcontrolador + Atmel ZibBit Amplified	Microcontrolador + Digi XBee ZB	Microcontrolador + Digi XBee ZB Pro
Wireless :	IEEE802.15.4	IEEE802.15.4	IEEE802.15.4	IEEE802.15.4
Emissor :	+3 dBm	+20 dBm	+3 dBm	+17 dBm
Receptor :	-101 dBm	-104 dBm	-96 dBm	-102 dBm
Alcance :	??	??	120 m	1600 m
Vantagens :	??	??	??	??
Desvantagens :	Precisa de microcontrolador	Precisa de microcontrolador e antena externa	Precisa de microcontrolador	Precisa de microcontrolador
Preço Unidade :	10,80 €	22,70 €	14,60 €	25 €
Preço Desenvolvimento :	636 €	636 €	208 €	208 €

Analisando as tabelas anteriores, pode-se facilmente comprovar que os dois módulos com melhores performances relativamente ao emissor e receptor são o Microchip MRF24J40MB e o Atmel ZibBit Amplified. Em comparação directa, o Microchip tem vantagem face ao Atmel no preço por unidade, custo de desenvolvimento e montagem pois não necessita de antena externa. A única desvantagem é a sensibilidade do receptor que tem menos 2dBm. Uma vez que o alcance anunciado é mais que suficiente, a escolha recai no Microchip MRF24J40MB. Uma vez que esta unidade só estaria disponível no final do ano transacto, iniciou-se o projecto utilizando a unidade rádio da Microchip MRF24J40MA que é baseado no mesmo integrado MRF24J40, mas que tem como diferença principal o módulo de antena, que não sendo amplificado anuncia um alcance de 120 metros ao invés dos 1200 metros anunciado pela componente MRF24J40MB.

A utilização destes dois módulos obriga a utilização de um microcontrolador da Microchip uma vez que as *stacks* ZigBee, MiWi e MiWiP2P só funciona com estes microcontroladores. Após analisar as soluções possíveis, foi decidido utilizar o Microchip 18F452 pela simples razão que cumpre os requisitos de comunicação de rede sem fios e este microcontrolador já é usado pela empresa Contelec noutros projectos, evitando deste modo, caso o projecto passe para a produção, de ter de armazenar outra componente diferente em stock.

Após a decisão de hardware, foi encomendado o kit de desenvolvimento da Microchip relativamente ao módulo de rádio. O kit é composto por duas placas denominadas *PICDEM Z Motherboard*, duas unidades MRF24J40MA e uma placa de análise de tráfego denominada *ZENA Network Analyser*.

Figura 3.1 – *PICDEM Z Motherboard* com unidade rádio MRF24J40MA

Figura 3.2 – *ZENA Network Analyser*

Para poder avançar com o projecto de desenvolvimento, mais componentes seriam necessárias: dois conjuntos de 5 *displays* de 7 segmentos, um para mostrar o "Preço por Hora" e o segundo para indicar o "Total a Pagar", seriam também necessários 3 interruptores de pressão, um que será para a função de "Memória/OK", outro para "Incremento" e outro para "Cursor"; Estes interruptores permitirão ao utilizador final programar a tarifa e visualizar o total acumulado. Finalmente, tendo em conta os requisitos funcionais, são também necessários quatro sensores de pressão *microswitch*, que terão como função a detecção do estado da gaveta: se aberta, fechada ou em alarme.

Tendo em conta que não existem pinos de entrada e saída livres no microcontrolador para interligar 10 displays de 7 segmentos, é necessário efectuar multiplexagem e utilizar um descodificador de displays. Uma vez mais, foi dada preferência a material já utilizado pela Contelec por motivos de logística e foi decidido utilizar o descodificador 74HCT42N.

3.1.2 - Software

O kit de demonstração PICDEM Z da Microchip fornece as stacks para as tecnologias ZigBee, MiWi e MiWi P2P, programadas em linguagem C, utilizando o compilador MCC C18 da Microchip. O ambiente de programação é o MPLAB e pertence também à Microchip.

De modo a poder comparar o alcance, velocidade de resposta e espaço de memória que necessitam, é possível compilar e experimentar as stacks nas placas do kit de desenvolvimento. Contudo, não é possível experimentar as topologias em árvore nem emalhadas uma vez que o kit apenas é composto por duas *motherboards*, não sendo então possível testar o sistema operacional utilizando uma unidade como *router*. A tabela seguinte apresenta os tamanhos de código utilizado por cada uma das tecnologias, assim como as topologias suportadas e o tipo de rede que admitem:

Tabela 3.3 — Comparação dos protocolos IEEE 802.15.4

	ZigBee	**MiWi**	**MiWi P2P**
Tamanho do Código:	Coordenador: 29 Kb Router: 30,7 Kb Dispositivo Final: 23,5 Kb	Coordenador: 7,1 Kb Dispositivo Final : 5,2 Kb	Coordenador: 3,2 Kb Dispositivo Final: 3,2 Kb
Topologias:	Estrela, do tipo árvore e emalhada	Estrela, do tipo árvore e emalhada	Estrela e P2P
Rede:	65 536 nós máximos infinitos saltos	1024 nós máximos infinitos saltos	1024 nós máximos em P2P ou 128 nós em árvore 4 saltos máximos

O tamanho de código da tecnologia ZigBee é significativamente maior que o que necessitam as tecnologias MiWi e MiWi P2P mas completamente justificável pelo tipo de rede que admite. Em comparação directa entre as tecnologias MiWi e MiWi P2P, a diferença do tamanho de código é facilmente explicada pelo simples facto de a tecnologia MiWi P2P não permitir a ligação de routers.

Atendendo aos requisitos iniciais do projecto, em que apenas é necessária a interligação de 20 contadores de tempo de bilhar com a unidade central que irá ligar ao computador, a solução MiWi P2P em ligação do tipo estrela é a mais adequada ao projecto.

Tendo em conta que o projecto necessitará de uma interface gráfica em computador, foi escolhido desenvolver a aplicação em *Visual Basic 6*, uma vez que já era uma programação conhecida e com bom suporte *web*.

De modo a perceber a comunicação entre o computador e as motherboards disponíveis no kit de desenvolvimento, foi utilizado o software *HyperTerminal,* disponível no Windows, que

apresenta as tramas recebidas e permite enviar as tramas de um modo directo, utilizando a porta série.

3.1.3 - Análise do Microcontrolador Microchip 18F452

O Microchip 18F452 é um microcontrolador com 32 Kbytes de memória de programa, 256 bytes de EEPROM e 1536 bytes de memória RAM. A sua frequência de funcionamento vai desde os 32 KHz até aos 25 MHz. Para o desenvolvimento do projecto, foi decidido trabalhar com a frequência de 20MHz por ser uma frequência elevada.

Este microcontrolador permite ligação de três pinos de interrupção externa, Timer0 que poderá ser temporizador ou contador de 8 ou 16bits com 8bits de *prescaler*, Timer1 e Timer3 que poderão ser temporizadores ou contadores de 16bits e Timer2 que poderá ser temporizador ou contador com um registo de período de 8bits, que entre outros, poderá ser utilizado para um PWM, sendo que é possível atribuir prioridades (*Low* e *High*) a cada interrupção.

A nível de comunicações, o 18F452 contém um módulo MSSP com dois modos de funcionamento: SPI por três fios, que será utilizado para interligar o microcontrolador e a unidade rádio MRF24J40M e I^2C com o modo *Master-Slave*. Este microcontrolador contém também um módulo USART que suporta comunicação RS-485 e RS 232, sendo este último o protocolo escolhido para comunicar entre o computador e o coordenador, e também contém um módulo PSP (*Parallel Slave Port*).

Analisando o microcontrolador pelo prisma da facilidade de programação, existe uma função com bastante utilidade no desenvolvimento de um projecto: programação on-board, denominada ICSP, em que efectua a programação do microcontrolador através de dois pinos(PGD e PDC), juntamente com o *Master Clear.*

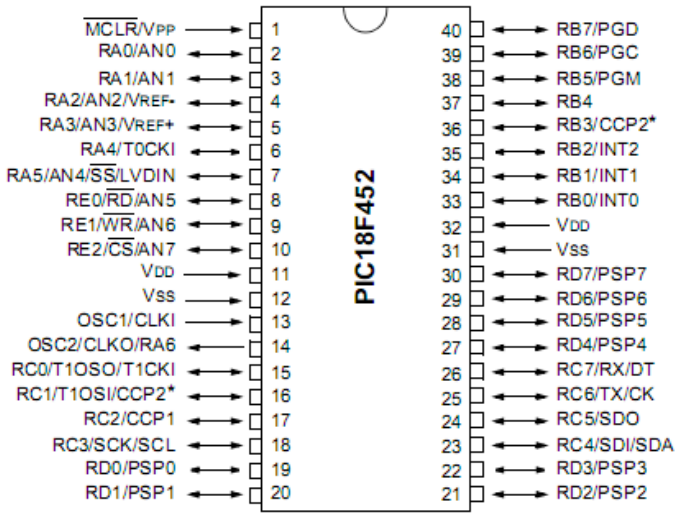
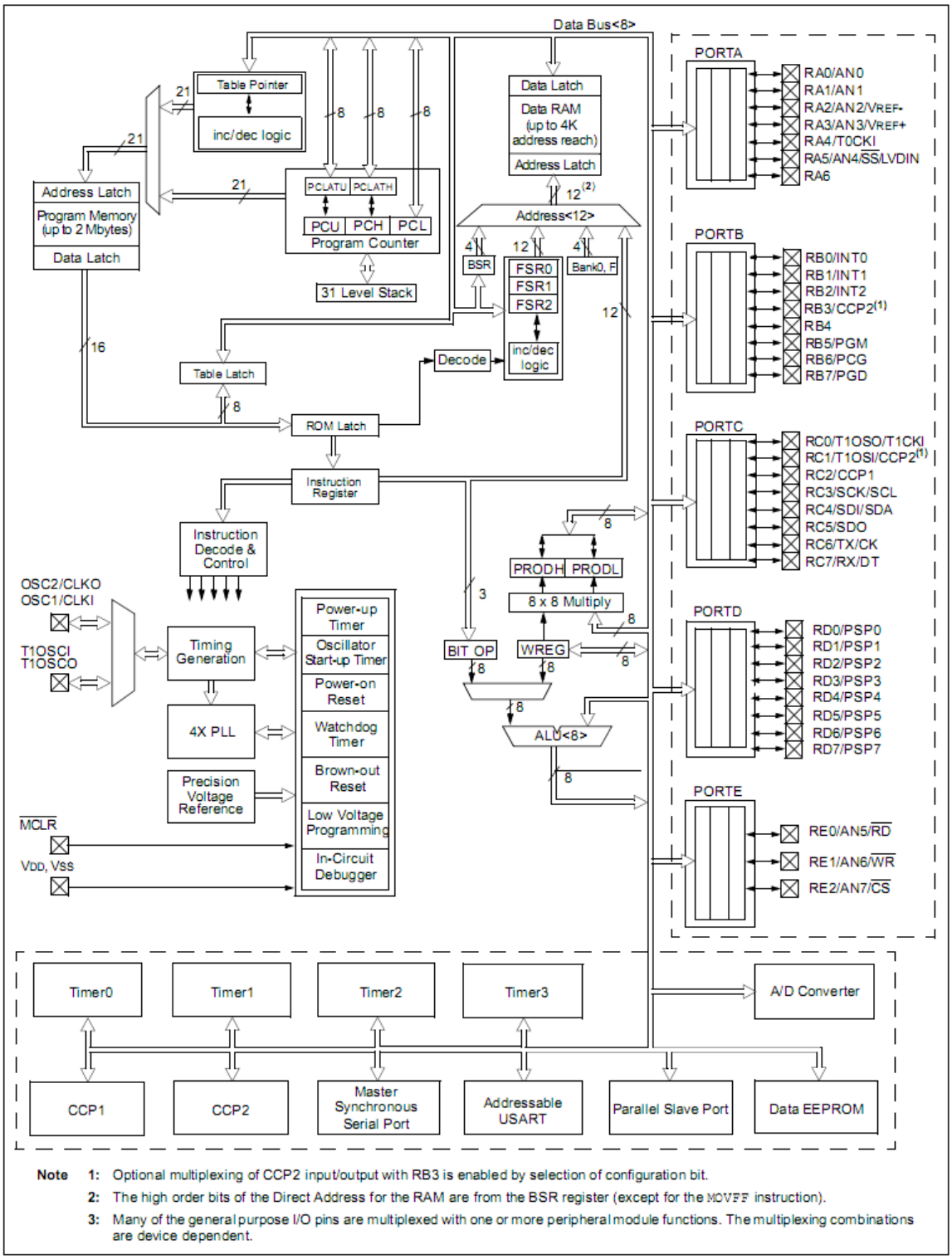


Figura 3.3 – *Pin-out* do microcontrolador Microchip 18F452 [19]



Note 1: Optional multiplexing of CCP2 input/output with RB3 is enabled by selection of configuration bit.
2: The high order bits of the Direct Address for the RAM are from the BSR register (except for the MOVFF instruction).
3: Many of the general purpose I/O pins are multiplexed with one or more peripheral module functions. The multiplexing combinations are device dependent.

Figura 3.4 - Diagrama de blocos do microcontrolador Microchip 18F452 [19]

3.1.3.1 - Organização da Memória

Existem três tipos de blocos de memória nos microcontroladores: Memória de Programa, Dados em RAM e Dados em EEPROM e poderá também haver Memória Flash. As memórias de dados e de programa utilizam barramentos diferentes.

3.1.3.1.1 - Memória de Programa

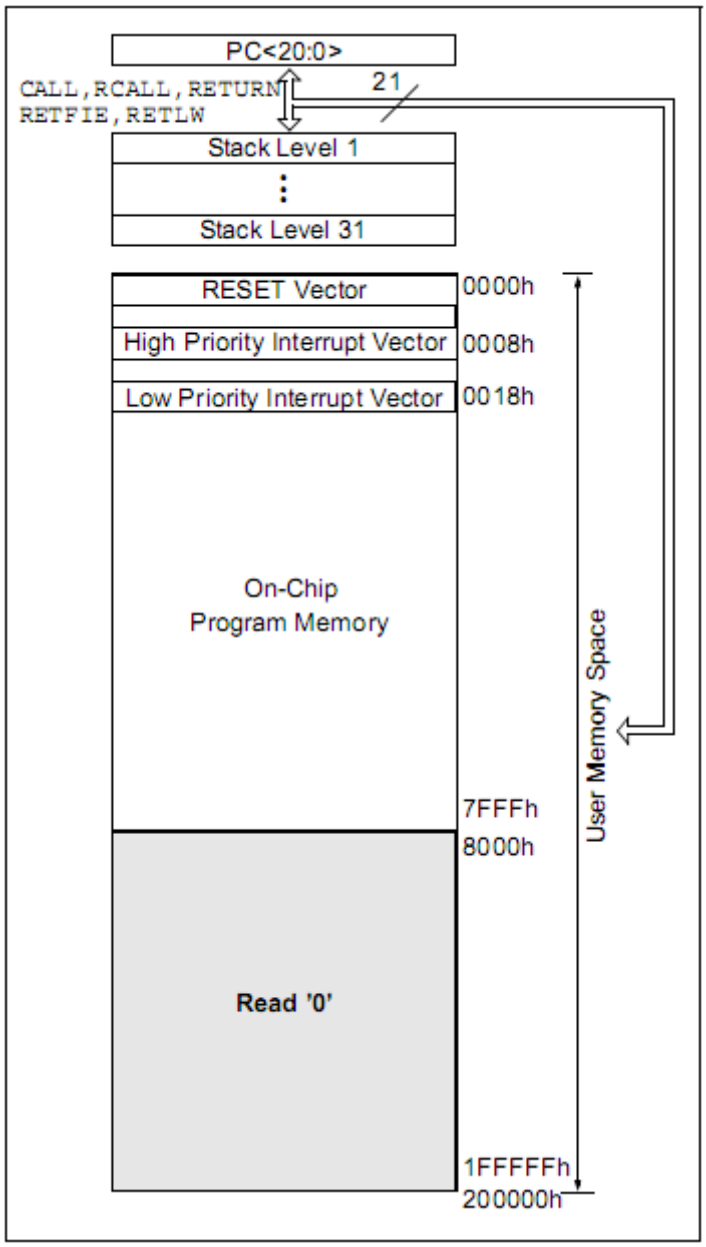


Figura 3.5 – Mapa da memória de programa e pilha do microcontrolador Microchip 18F452 [19]

Um contador de 21bits é capaz de endereçar até 2Mbytes de espaço de memória de programa. Ao aceder às posições de memória que se localizam entre a memória física implementada e os 2Mbytes irá causar uma leitura a zeros (instrução NOP). Este microcontrolador tem 32Kbytes de memória flash. Isto significa que este microcontrolador consegue guardar instruções até 16K. O endereço de reset é 0000h e os endereços de interrupção são 0008h e 0018h para as interrupções de baixa e alta prioridade, respectivamente.

3.1.3.1.2 - Memória EEPROM

A memória de dados da EEPROM permite efectuar leitura e escrita durante o funcionamento normal do microcontrolador. A memória de dados não é directamente mapeada no registo, mas é endereçada através de uns registos especiais denominados SFRs.

Existem quatro SFRs que são usados para ler e guardar valores na EEPROM: EECON1, EECON2, EEDATA e EEADR.

Quando em contacto com o bloco da memória de dados, EEDATA mantém o valor de 8 bits para leitura/escrita e o EEADR mantém a posição a ser endereçada. Este dispositivo tem 256 bytes de EEPROM que vão desde a posição 0h até à FFh. EECON1 é o registo de controlo para o acesso à memória. EECON2 não é um registo físico. Este registo é apenas usado na sequência de gravação de dados.

Esta memória é destinada a elevados ciclos de apagar/escrever. Um byte escrito automaticamente apaga o que se encontrava na mesma posição e posteriormente irá ser procedida a gravação.

Para ler da memória, o utilizador deve gravar a posição pretendida no registo EEADR, limpar os bits EEPGD e CFGS do registo EECON1 e definir o bit RD do mesmo registo. No próximo ciclo, a data estará disponível no registo EEDATA.

Para gravar na memória, tal como na leitura, o utilizador começa por gravar a posição pretendida no registo EEADR e gravar os dados no registo EEDATA. Por questões de segurança, de seguida é necessário gravar o valor 55h ao registo EECON2 e no passo seguinte gravar o valor AAh ao mesmo registo. Este passo serve para evitar gravações acidentais. Finalmente, define-se o bit WR no registo EECON1.

3.1.3.2 - Interrupções

Este microcontrolador tem múltiplas fontes de interrupções e com prioridades entre elas, que permitem ser atribuídas prioridade alta ou prioridade baixa. O vector de uma interrupção de prioridade alta é a 000008h enquanto o vector de uma prioridade de baixa prioridade é a 000018h.

As interrupções de alta prioridade fazem um *override* às prioridades baixas em progresso.

Existem dez registos que são usados para controlar as interrupções: RCON, INTCON, INTCON2, INTCON3, PIR1, PIR2, PIE1, PIE2, IPR1 e IPR2.

Cada fonte de interrupção excepto o INT0 tem três bits para controlar as suas operações. As funções desses três bits são: uma *flag* que indica que uma interrupção ocorreu, um bit de activação que permite que o programa execute o código associado à interrupção quando a flag assim o indica e um bit de prioridade que pode tomar o valor de *high* e *low*.

A prioridade das interrupções é activada quando se define o bit IPEN do registo RCON. Quando a prioridade está activa, existem dois bits que activam globalmente as interrupções: GIEH e GIEL, ambos do registo INTCON, em que o primeiro activa as interrupções que têm o bit de prioridade activado e o segundo activa as que têm o bit de prioridade desactivado.

O bit PEIE activa ou desactiva todas as fontes de interrupções externas. O bit GIE activa ou desactiva todas as fontes de interrupções.

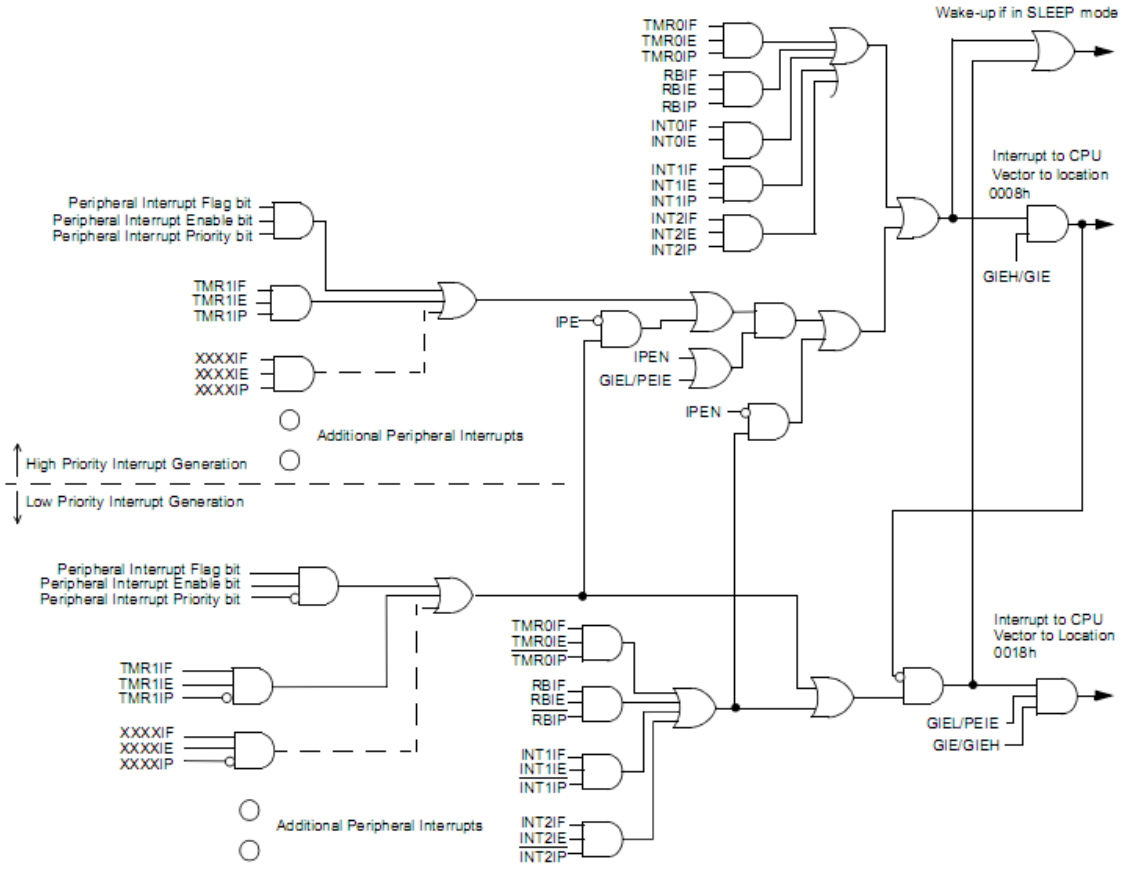


Figura 3.6 – Lógica das Interrupções [19]

Os registos PIR contêm *flags* individuais para as interrupções periféricas. Devido ao número de interrupções periféricas, existem dois registos: PIR1 e PIR2. Os registos PIE contêm os bits de activação das interrupções periféricas. Mais uma vez existem dois registos (PIE1 e PIE2) devido ao número de interrupções periféricas. Relativamente às interrupções periféricas existem os registos IPR1 e IPR2 que contém os bits relativos às prioridades destas interrupções.

O registo RCON contém os bits que são utilizados para activar as prioridades com prioridade.

As interrupções externas nos pinos RB0/INT0, RB1/INT1 e RB2/INT2 são activadas na mudança de estado. Ou quando passam de 0 para 1, caso o bit INTEDGx esteja activado no registo INTCON2, ou de 1 para 0, caso o mesmo bit esteja a zero. Quando essa mudança é detectada, a flag INTxF é imediatamente definida. A interrupção pode ser desactivada ao limpar o bit INTxE correspondente. A flag deve ser limpa manualmente após a execução do código da interrupção de modo a poder ser detectada nova interrupção.

3.1.3.3 - Portas I/O

Existem cinco blocos de portas de entrada e saída disponíveis. Alguns pinos das portas de E/S estão multiplexados com diferentes funções relacionadas com periféricos. De um modo geral, quando um pino é utilizado para controlar um periférico, já não pode ser utilizado como pino de entrada ou saída.

Cada porta tem três registos: TRIS (registo relativo ao sentido da porta: entrada ou saída), PORT (lê os níveis dos pinos: 0 ou 1) e LAT (*latch* de saída). Este último registo é utilizado para operações de leitura e consequente alteração do valor.

3.1.3.3.1 - PORTA, TRISA e LATA

PORTA é uma porta de 7 bits bidireccionais. Caso TRISA esteja definida (=1), a PORTA será de entrada. Caso TRISA esteja limpa (=0), a PORTA será de saída.

O pino RA4 está multiplexado com a entrada do *clock* do módulo do timer0. Este pino, RA4/T0CKI é uma entrada s*chmitt trigger* e uma saída *open drain*.

Todos os outros pinos estão multiplexados com entradas analógicas e com entradas V_{REF+} e V_{REF-}.

Ao efectuar um reset, RA5 e RA3:RA0 estão definidas com entradas analógicas e RA6 e RA4 estão definidas como entradas digitais.

Tabela 3.4 — Funções da PORTA [19]

Bit	Nome	Funções
0	RA0/AN0	Pino E/S ou entrada analógica
1	RA1/AN1	Pino E/S ou entrada analógica
2	RA2/AN2/V_{REF-}	Pino E/S ou entrada analógica ou V_{REF-}
3	RA3/AN3/V_{REF+}	Pino E/S ou entrada analógica ou V_{REF+}
4	RA4/T0CKI	Pino E/S ou entrada de clock externa para o Timer0. Saída é do tipo *open drain*
5	RA5/~~SS~~/AN4/LVDIN	Pino E/S ou entrada de escolha de *slave* para porta série síncrona ou entrada analógica ou detecção de baixa tensão
6	RA6/OSC2/CLKO	Pino E/S ou OSC2 ou saída de *clock*

3.1.3.3.2 - PORTB, TRISB e LATB

PORTB é uma porta de 8 bits bidireccionais. Caso TRISB esteja definida (=1), a PORTB será de entrada. Caso TRISB esteja limpa (=0), a PORTB será de saída.

Cada um dos pinos da PORTB tem um *pull-up* interno fraco. Ao limpar o bit ~~RBPU~~ do registo INTCON2, esses *pull-ups* são activados. Os *pull-ups* são desactivados quando os pinos estão configurados como saídas e são desactivados sempre que acontece um reset. Ao efectuar um reset, os pinos são configurados como entradas digitais.

Quatro dos pinos da PORTB, RB7:RB4 permitem gerar interrupção com a mudança de estado, apenas caso estejam configurados como entrada. Esta interrupção pode acordar o dispositivo do modo *sleep*. O pino RB5 está multiplexado com o pino de activação de programação *in-circuit* de baixa voltagem; o pino RB6 está multiplexado com o *clock* de programação *in-circuit* e o pino RB7 está multiplexado com os dados da programação *in-circuit*.

Tabela 3.5 — Funções da PORTB [19]

Bit	Nome	Funções
0	RB0/INT0	Pino E/S ou entrada de interrupção externa
1	RB1/INT1	Pino E/S ou entrada de interrupção externa
2	RB2/INT2	Pino E/S ou entrada de interrupção externa
3	RB3/CCP2	Pino E/S ou entrada de Captura2 ou saída de Comparação2 ou saída de PWM quando o bit CCP2MX está activo
4	RB4	Pino E/S (com *interrupt on change*)
5	RB5/PGM	Pino E/S (com *interrupt on change*) Pino de activação de programação *in-circuit* com baixa tensão
6	RB6/PGC	Pino E/S (com *interrupt on change*) *Clock* de programação série
7	RB7/PGD	Pino E/S (com *interrupt on change*) Dados de programação série

3.1.3.3.3 - PORTC, TRISC e LATC

PORTC é uma porta de 8 bits bidireccionais. Caso TRISC esteja definida (=1), a PORTC será de entrada. Caso TRISC esteja limpa (=0), a PORTC será de saída.

A PORTC está multiplexada com várias funções relativas a dispositivos periféricos. Os seus pinos têm um *buffer* de entrada *schmitt trigger*.

Ao activar as funções relativas a periféricos, há que definir TRISC para cada pino pois alguns dispositivos fazem um *override* ao TRIS para fazerem com que um pino seja entrada e outros fazem com que seja saída.

Ao efectuar um reset, estes pinos são definidos como entradas digitais.

Tabela 3.6 — Funções da PORTC [19]

Bit	Nome	Funções
0	RC0/T1OSO/T1CKI	Pino E/S ou saída do oscilador do Timer1 ou entrada do clock do Timer1
1	RC1/T1OSI/CCP2	Pino E/S ou entrada do oscilador do Timer1 ou entrada de Captura2 ou Saída do Comparador2 ou saída de PWM quando o bit CCP2MX está activo
2	RC2/CCP1	Pino E/S ou entrada de Captura1 ou saída de Comparador1 ou saída de PWM1
3	RC3/SCK/SCL	Pino E/S ou clock de SPI ou clock de I^2C
4	RC4/SDI/SDA	Pino E/S ou data de SPI ou data de I^2C
5	RC5/SDO	Pino E/S ou saída de dados síncrona da porta série.
6	RC6/TX/CK	Pino E/S ou emissor assíncrono da USART ou clock síncrono da USART
7	RC7/RX/DT	Pino E/S ou receptor assíncrono da USART ou dados síncronos da USART

3.1.3.3.4 - PORTD, TRISD e LATD

PORTD é uma porta de 8 bits bidireccionais. Caso TRISD esteja definida (=1), a PORTD será de entrada. Caso TRISD esteja limpa (=0), a PORTD será de saída.

A PORTD é uma porta de 8 bits com uma entrada schmitt trigger. Cada pino é individualmente configurável como entrada ou saída. Ao efectuar um reset, estes pinos são configurados como entradas digitais. Esta porta pode ser configurada como uma porta de microprocessador com 8 bits de largura (porta paralela), ao definir o bit PSPMODE no registo TRISE. Neste modo os *buffers* de entrada são do tipo TTL.

Tabela 3.7 — Funções da PORTD [19]

Bit	Nome	Funções
0	RD0/PSP0	Pino E/S ou bit0 da porta paralela *slave*
1	RD1/PSP1	Pino E/S ou bit1 da porta paralela *slave*
2	RD2/PSP2	Pino E/S ou bit2 da porta paralela *slave*
3	RD3/PSP3	Pino E/S ou bit3 da porta paralela *slave*
4	RD4/PSP4	Pino E/S ou bit4 da porta paralela *slave*
5	RD5/PSP5	Pino E/S ou bit5 da porta paralela *slave*
6	RD6/PSP6	Pino E/S ou bit6 da porta paralela *slave*
7	RD7/PSP7	Pino E/S ou bit7 da porta paralela *slave*

3.1.3.3.5 - PORTE, TRISE e LATE

PORTE é uma porta de 3 bits bidireccionais. Caso TRISE esteja definida (=1), a PORTE será de entrada. Caso TRISE esteja limpa (=0), a PORTE será de saída.

Esta porta tem três pinos que são individualmente configuráveis como entrada ou saída. Estes pinos têm *buffers* de entrada do tipo *schmitt trigger*.

Os pinos da PORTE estão multiplexados com entradas analógicas. Mesmo quando o utilizador usar estes pinos como entradas analógicas, deve certificar-se que o TRISE está configurado para entrada.

Sempre que houver um reset, estes pinos estão definidos como entradas analógicas.

Tabela 3.8 — Funções da PORTE [19]

Bit	Nome	Funções
0	RE0/~~RD~~/AN5	Pino E/S ou controlo de leitura da porta paralela ou entrada analógica: RD 1=não é operação de leitura 0=Operação de leitura. Lê o registo PORTD (caso o dispositivo esteja escolhido)
1	RE1/~~WR~~/AN6	Pino E/S ou controlo de escrita da porta paralela ou entrada analógica: WR 1=não é operação de escrita 0=Operação de escrita. Escreve no registo PORTD (caso o dispositivo esteja escolhido)
2	RE2/~~CS~~/AN7	Pino E/S ou de escolha de dispositivo da porta paralela ou entrada analógica: CS 1=O dispositivo não está seleccionado 0=O dispositivo está seleccionado

3.1.3.4 - Timers

Este microcontrolador tem quatro *timers* internos: Timer0, Timer1, Timer2 e Timer3.

3.1.3.4.1 - Timer0

O módulo Timer0 pode funcionar como um *timer* ou contador de 8bits ou a 16bits, permite leitura e escrita, contém um *prescaler* programável de 8bits, permite escolher entre a utilização de um *clock* interno ou externo e efectua uma interrupção ao ocorrer *overflow*.

O modo timer é definido ao limpar o bit T0CS. Caso T0CS esteja activo, o modo contador será utilizado.

No modo timer, o módulo irá incrementar a todos os ciclos (com o *prescaler* desactivado). No modo contador, o módulo irá incrementar sempre que houver mudança de estado. Caso o bit T0SE seja limpo, o módulo irá incrementar quando o pino RA4/T0CKI passar de zero a um. Caso esteja definido, o módulo irá incrementar quando o pino anterior passar de um a zero.

O bit PSA do registo T0CON activa ou desactiva o *prescaler* e os bits T0PS2:T0PS0 irão determinar a escala do mesmo. Ao escrever no bit TMR0L, o valor de *prescaler* será reiniciado. Sempre que houver *overflow*, uma interrupção será gerada e o bit TMR0IF é activado. As interrupções do timer podem ser activadas ou desactivas consoante o bit TMR0IE. As interrupções dos timers não acordam o dispositivo do modo *sleep*.

O byte mais elevado do Timer0 não permite ser lido ou escrito directamente. TMR0H é actualizado com os conteúdos do byte mais elevado quando o bit TMR0L é lido. Isto permite ler todos os 16 bytes do Timer0 sem ter que verificar se as leituras do TMR0L e TMR0H são válidas, isto é, se não houve *rollover* entre as leituras sucessivas.

3.1.3.4.2 - Timer1

O módulo Timer1 pode funcionar como um *timer* ou contador (síncrono ou assíncrono) de 8bits ou a 16bits, permite leitura e escrita, contém um *prescaler* programável de 3bits, permite escolher entre a utilização de um *clock* interno ou externo e efectua uma interrupção ao ocorrer *overflow*.

O modo de operação é definido pelo bit TMR1CS do registo T1CON. Caso TMR1CS seja 0, o Timer1 incrementa a cada ciclo de instruções; caso seja 1, o Timer1 incrementa sempre que o *clock* externo passar de 0 a 1. Caso o oscilador esteja activo, os pinos RC1/T1OSI e RC0/T1OSO/T1CKI são considerados pinos de entrada.

A interrupção TMR1, caso esteja activada, é activada por *overflow*, activando o bit TMR1IF. Esta interrupção pode ser activada ou desactivada através do bit TMR1IE.

O Timer1 pode ser configurado para ler e escrever a 16bits, caso o bit RD16 do registo T1CON esteja activo.

3.1.3.4.3 - Timer2

O módulo Timer2 pode funcionar como um *timer* ou registo de períodos de 8bits, permite leitura e escrita, contém um *prescaler* programável de 4bits, contém um *postscaler* de 4 bits e efectua uma interrupção ao ocorrer *overflow*. Este *timer* pode ser utilizado como base de tempo para um PWM. O registo TMR2 permite ser lido e escrito e é reiniciado sempre que existe um reset de um dispositivo.

Os contadores de *prescaler* e *postscaler* são reiniciados sempre que se dá uma gravação no registo TMR2 ou no T2CON ou sempre que houver um reset de um dispositivo. O valor de TOMR2 não é reiniciado quando houver uma escrita no registo T2CON.

3.1.3.4.4 - Timer3

O módulo Timer3 pode funcionar como um *timer* ou contador (síncrono ou assíncrono) de 16bits, permite leitura e escrita, contém um *prescaler* programável de 3bits, permite escolher entre a utilização de um *clock* interno ou externo e efectua uma interrupção ao ocorrer *overflow*.

3.1.3.5 - Comunicação MSSP

O módulo MSSP é uma interface série bastante útil para comunicações com outros periféricos ou microcontroladores. Estes periféricos podem ser vários, desde a EEPROMs, registos *shift*, controladores de *displays*, conversores A/D, unidades de rádio, etc.

Este módulo pode funcionar em dois módulos: SPI e I^2C.

3.1.3.5.1 - SPI

O módulo SPI permite que 8bits de dados sejam transmitidos e recebidos de um modo síncrono e simultaneamente. De modo a permitir a comunicação, normalmente são utilizados 3 pinos: *Serial Data Output* (SDO), *Serial Data Input* (SDI) *e Serial Clock* (SCK). Adicionalmente um outro pino poderá ser utilizado quando utilizado em modo *slave*: *Slave Selection* (~~SS~~).

O módulo MSSSP contém quatro registos para o modo de operação SPI: SSPCON1 (*MSSP Control Register1*), SSPSTAT (*MSSP Status Register*), SSPBUF (*Serial Receive/Transmit Buffer*) e SSPSR (*MSSP Shift Register*). Os registos SSPCON1 e SSPSTAT são os registos de controlo e status do modo de operação SPI.

Ao inicializar o SPI, várias opções têm de ser configuradas utilizando os registos anteriores SSPCON1 e SSPSTAT. Estes registos permitem definir: escolha de modo *master* ou *slave*, polaridade do *clock*, fase de amostragem de dados de entrada, *clock edge* (saída de dados quando o relógio passa a 1 ou quando passa a 0), *clock rate* (para o modo *master*) e *slave select* (para o modo *slave*).

Para activar o SPI é necessário activar o bit SSPEN do registo SSPCON1. Ao efectuar esta activação, os pinos SDI, SDO, SCK e ~~SS~~ são automaticamente configurados para serem usados na comunicação série, apesar de terem de estar devidamente configurados no registo TRIS.

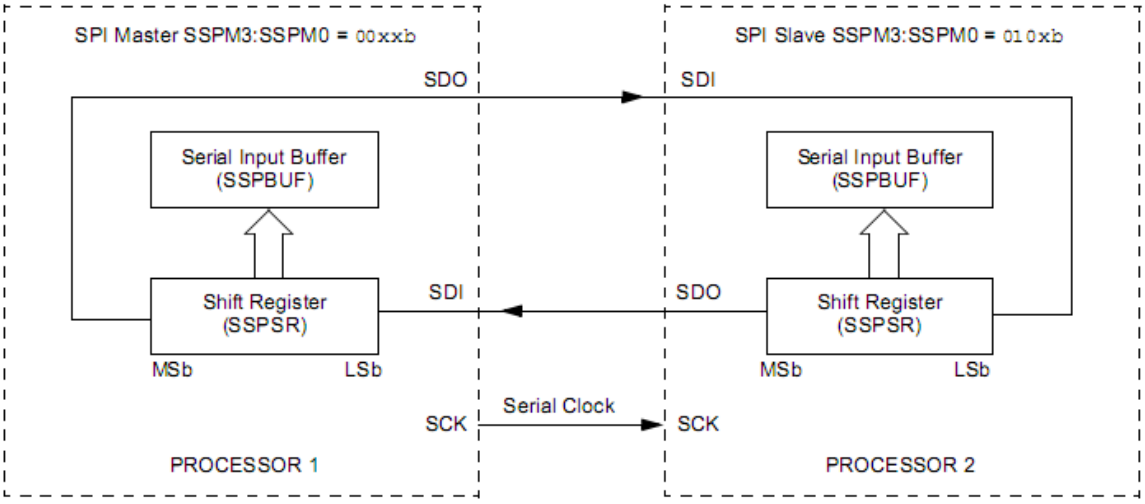


Figura 3.7 – Ligação típica do modo SPI [19]

Caso o sistema esteja definido no modo master, este pode iniciar a comunicação a qualquer momento pois é ele que controla o SCK e também determina quando o *slave* envia informação.

Neste modo, os dados são enviados/recebidos à medida que o registo SSPBUF é escrito.

Caso o sistema esteja definido no modo slave, a informação é enviada e recebida à medida que o um impulso do *clock* externo aparece no SCK.

Quando o sistema se encontra em modo *sleep*, o *slave* pode receber ou enviar dados. O dispositivo irá acordar quando um byte for recebido.

3.1.3.5.2 - I^2C

O módulo MSSP no modo I^2C implementa na totalidade todas as funções de *master* e de *slave* e fornece interrupções nos bits de inicio e de fim no hardware para determinar o barramento livre. Este módulo implementa as especificações *standard* assim como endereçamentos de 7bits e de 10bits.

Os dois pinos utilizados para comunicação são o SCL (*Serial Clock*) e o SDA (*Serial Data*). Estes pinos devem estar configurados como entradas e saídas no TRISC.

O módulo MSSP no modo I^2C tem seis registos: SSPCON1 (MSSP Control Register1), SSPCON2 (MSSP Control Register2), SSPSTAT (MSSP Status Register), SSPBUF (Serial Receive/Transmit Buffer), SSPSR (MSSP Shift Register) e SSPADD (MSSP Address Register). Os três primeiros são os registos de controlo e de estado.

As funções do módulo MSSP activam-se ao definir o bit SSPEN do registo SSPCON. O registo SSPCON1 permite escolher o modo de funcionamento do I^2C: *Master Mode*, *Slave Mode* (com 7 bits de endereçamento), *Slave Mode* (com 10 bits de endereçamento), *Slave Mode* (com 7 bits de endereçamento e interrupção de início e de fim), *Slave Mode* (com 10 bits de endereçamento e interrupção de início e de fim) e finalmente operações de *master* controladas por *firmware*.

No modo *slave*, quando o endereço coincidir e quando a transferência de dados após a coincidência do endereço estiver concluída, o hardware automaticamente irá gerar um impulso ~~ACK~~ (*Aknowledge*) e irá carregar os dados no registo SSPBUF.

O modo *master* é suportado no facto de gerar interrupções na detecção de início e de fim.

Quando o módulo MSSP está configurado no modo *master* I2C, não permite filas de espera de eventos. Por exemplo, não é permitido ao utilizador iniciar uma condição de início e imediatamente escrever no SSPBUF sem que a condição de início esteja completa, perdendo desse modo o valor escrito no SSPBUF.

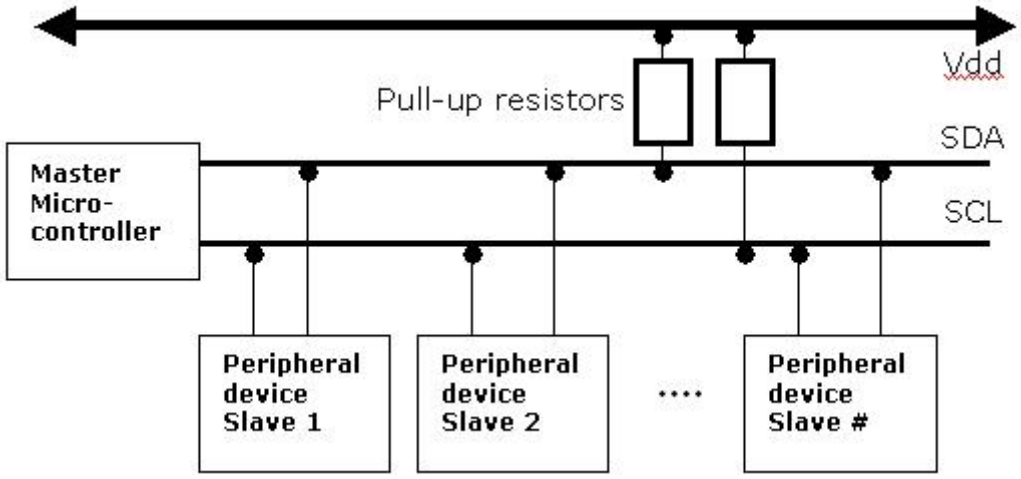


Figura 3.8 – Ligação típica do modo I^2C

3.1.3.6 - Comunicação USART

O módulo USART (*Universal Synchronous Asynchronous Receiver Transmitter*) é um dos dois módulos de comunicação série. A USART pode ser configurada como um sistema assíncrono *full duplex* que pode comunicar com terminais CRT e computadores pessoais, ou pode ser configurado como sistema síncrono *half duplex* em que pode comunicar com periféricos tais como conversores A/D e D/A, EEPROMs, etc.

O BRG (BaudRate Generator) suporta tanto o modo síncrono como o modo assíncrono e é um gerador de baud rate de 8 bits dedicado. O registo SPBRG controla o período do funcionamento livre do timer de 8bits. No modo assíncrono, o bit BRGH controla o baud rate. No modo síncrono, este bit é ignorado.

3.1.3.6.1 - Erro de BaudRate

Para calcular o erro de baud rate, utiliza-se a seguinte fórmula:
Baud Rate Desejado = FOSC / (64 (X-1))
Sendo que o X corresponde ao valor SPBRG e varia entre 0 e 255.

Após calcular o valor de X, calcula-se o baud rate calculado e calcula-se o erro:
Erro = (Baud Rate Calculado – Baud Rate Desejado) / Baud Rate Desejado

3.1.3.6.2 - USART Assíncrono

Neste modo, a USART utiliza o modo formato standard NRZ que consiste no bit de início, oito ou nove bits de dados e o bit de fim de trama. A USART envia e recebe o LSB primeiro. O emissor e receptor são independentes entre si, mas utilizam o mesmo baud rate e o mesmo formato de dados. A paridade não é suportada por hardware, mas poderá ser implementada em software.

3.1.3.6.3 - USART Síncrono no Modo *Master*

No modo síncrono, os dados são transmitidos no modo half-duplex, ou seja, a recepção e o envio não podem ocorrer ao mesmo tempo. O modo master apenas indica que o processador transmite o master clock no pino CK, sendo este gerado internamente.

3.1.3.6.4 - USART Síncrono no Modo *Slave*

Este modo difere do anterior apenas no aspecto do clock ser fornecido pelo master através do pino RC6/TX/CK. Esta particularidade permite que o dispositivo receba ou envie dados estando no modo sleep.

3.1.4 - Análise do Módulo RF MRF24J40MA

Este dispositivo é uma unidade rádio que opera na banda não licenciada dos 2,4 GHz, e pertence às recomendações IEEE 802.15.4. O módulo MRF24J40MA trata-se de um circuito impresso PCB que contém o circuito integrado MRF24J40, um cristal de 20MHz e todas as componentes necessárias para o seu funcionamento. A antena encontra-se incluída no PCB, facilitando a sua instalação.

Este módulo suporta os protocolos ZigBee, MiWi e MiWi P2P, sendo que cada uma das *stacks* se encontra disponível no site da Microchip. O modo de interligação entre o módulo e o microcontrolador é por comunicação SPI, obrigando à utilização em conjunto de um microcontrolador da Microchip de modo a que as stacks funcionem correctamente. O alcance anunciado pelo fabricante é de apenas 100m em campo aberto, reduzindo para cerca de 1/3 *indoors*. Este alcance é reduzido e limita a versatilidade de todo o sistema, mas não é um factor crítico uma vez que este não será o módulo de comunicação utilizado em produção. A figura seguinte representa o pin-out do módulo MRF24J40MA, que é compatível pino a pino com o MRF24J40MB, sendo que o segundo é o módulo de maior alcance e a ser utilizado em produção.

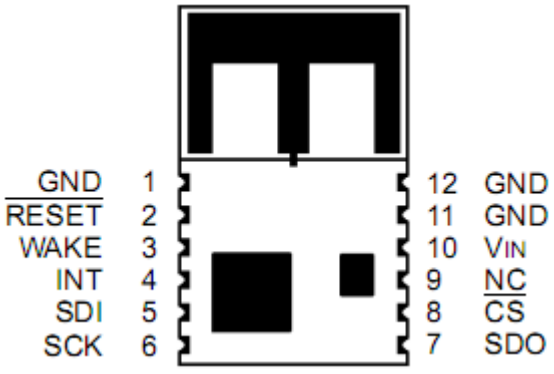


Figura 3.9 – Pin-out do MRF24J40MA

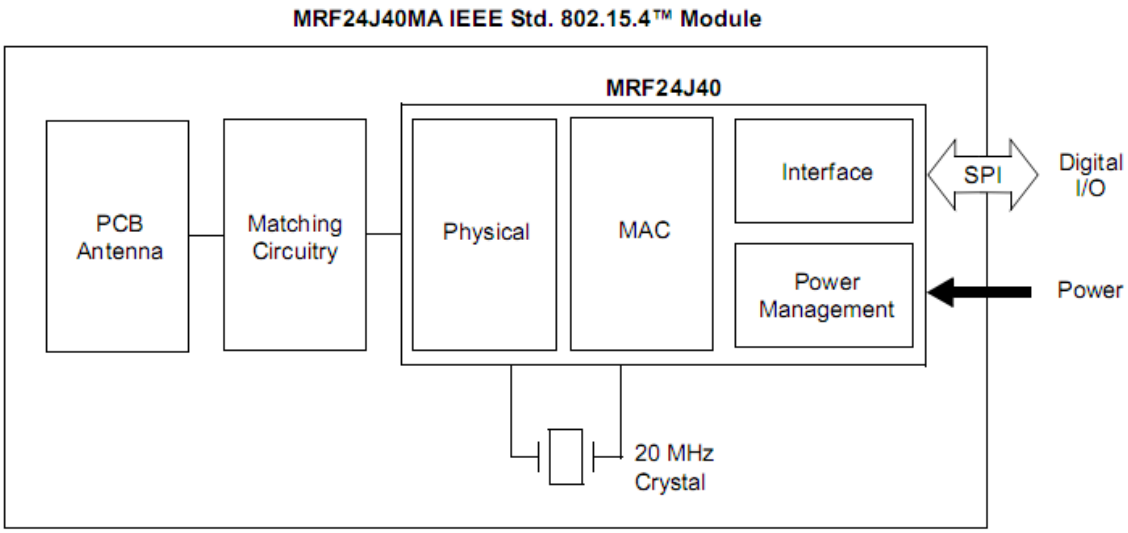


Figura 3.10 – Diagrama de Blocos do MRF24J40MA

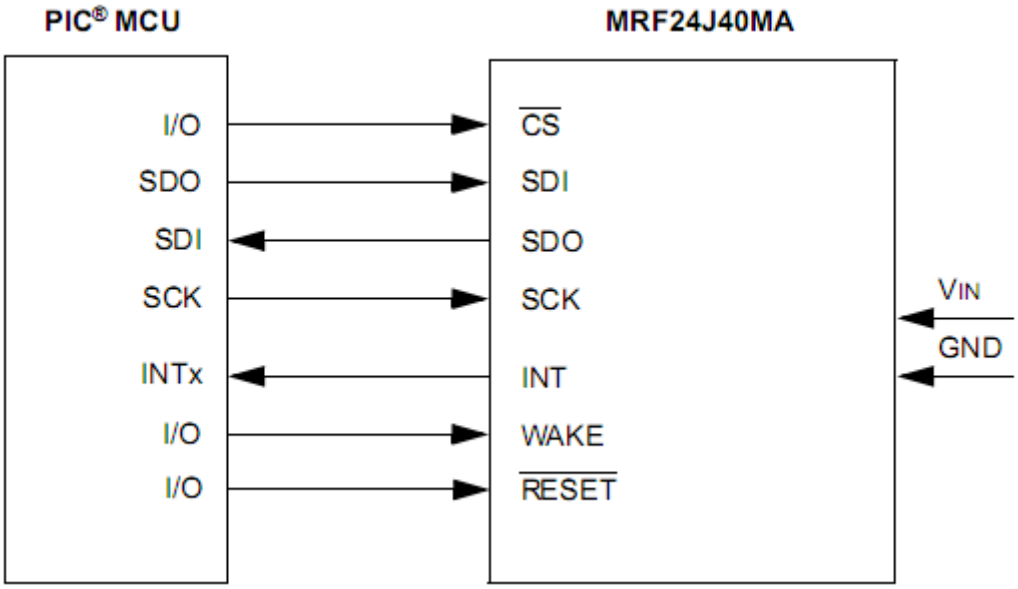


Figura 3.11 – Interligação entre o microcontrolador e o módulo MRF24J40MA

3.2 - Projecto e Desenvolvimento

Após efectuar as análises comparativas entre as tecnologias de hardware e as tecnologias de rede sem fios disponíveis, assim como a análise do microcontrolador escolhido e módulo de rádio para a comunicação sem fios, pode-se avançar para o projecto e desenvolvimento.

Este subcapitulo está dividido em duas partes distintas: desenvolvimento físico e desenvolvimento de programação. No desenvolvimento físico, é analisado o circuito de montagem, efectuada a sua esquematização, desenho e produção do PCB e finalmente a sua montagem. Relativamente ao desenvolvimento de programação, é esquematizado o princípio de funcionamento da motherboard de

coordenador, assim como esquematizado o princípio de funcionamento do circuito impresso produzido e apresentada a aplicação em Visual Basic que monitoriza o estado da rede de contadores de tempo de bilhar, através da comunicação série existente entre o computador e a motherboard de coordenador e através da comunicação sem fios entre a motherboard e as placas produzidas para o contador de tempo de bilhar.

3.2.1 - Esquema e Desenho do circuito PCB

Tendo em conta os requisitos iniciais, verifica-se que é necessário interligar dez displays de 7 dígitos, três interruptores e quatro sensores de pressão. Uma vez que cada *display* necessita de oito pinos, seriam necessários 80 pinos de entrada e saída só para controlar os displays, situação impossível utilizando este microcontrolador. Estando o número de pinos de entrada e saída limitados, a solução passa pela multiplexagem e pela utilização de um descodificador para a escolha do display a acender.

3.2.1.1 - Esquema dos *Displays*

Uma vez que a Contelec já utilizava descodificadores para os displays nos seus projectos actuais, foi escolhido foi o descodificador 74HCT42N que permite, com uma entrada de 4 bits, controlar até 10 displays, poupando deste modo 6 pinos de entrada/saída no microcontrolador. O *pin-out* do descodificador pode ser observado na figura seguinte:

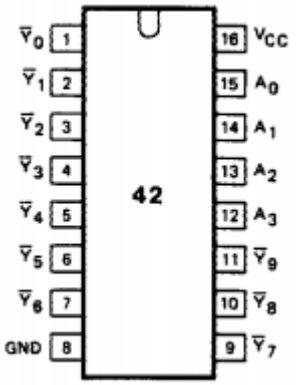


Figura 3.12 – Pin-out do descodificador 74HCT42N

A resposta do descodificador em função das entradas é mostrada na tabela seguinte. Teoricamente com 4 bits de entrada é possível controlar 16 saídas, situação que não acontece com este descodificador uma vez que apenas contém 16 pinos, sendo que 4 são de entrada e 2 são de alimentação, restando apenas 10 pinos para saídas.

Tabela 3.9 — Tabela de resposta do descodificador

Entradas				Saídas									
A_3	A_2	A_1	A_0	$\overline{Y}_0$	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	$\overline{Y}_4$	$\overline{Y}_5$	$\overline{Y}_6$	$\overline{Y}_7$	$\overline{Y}_8$	$\overline{Y}_9$
L	L	L	L	L	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	H	H	L	H	H	H	H	H	H	H
L	L	H	H	H	H	H	L	H	H	H	H	H	H
L	H	L	L	H	H	H	H	L	H	H	H	H	H
L	H	L	H	H	H	H	H	H	L	H	H	H	H
L	H	H	L	H	H	H	H	H	H	L	H	H	H
L	H	H	H	H	H	H	H	H	H	H	L	H	H
H	L	L	L	H	H	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L
H	L	H	L	H	H	H	H	H	H	H	H	H	H
H	L	H	H	H	H	H	H	H	H	H	H	H	H
H	H	L	L	H	H	H	H	H	H	H	H	H	H
H	H	L	H	H	H	H	H	H	H	H	H	H	H
H	H	H	L	H	H	H	H	H	H	H	H	H	H
H	H	H	H	H	H	H	H	H	H	H	H	H	H

As ligações entre o descodificador e os dois conjuntos de displays poderão ser observadas na figura seguinte, tendo em conta que as ligações "DAB" referem-se aos 4 bits de ligação entre o microcontrolador e o descodificador.

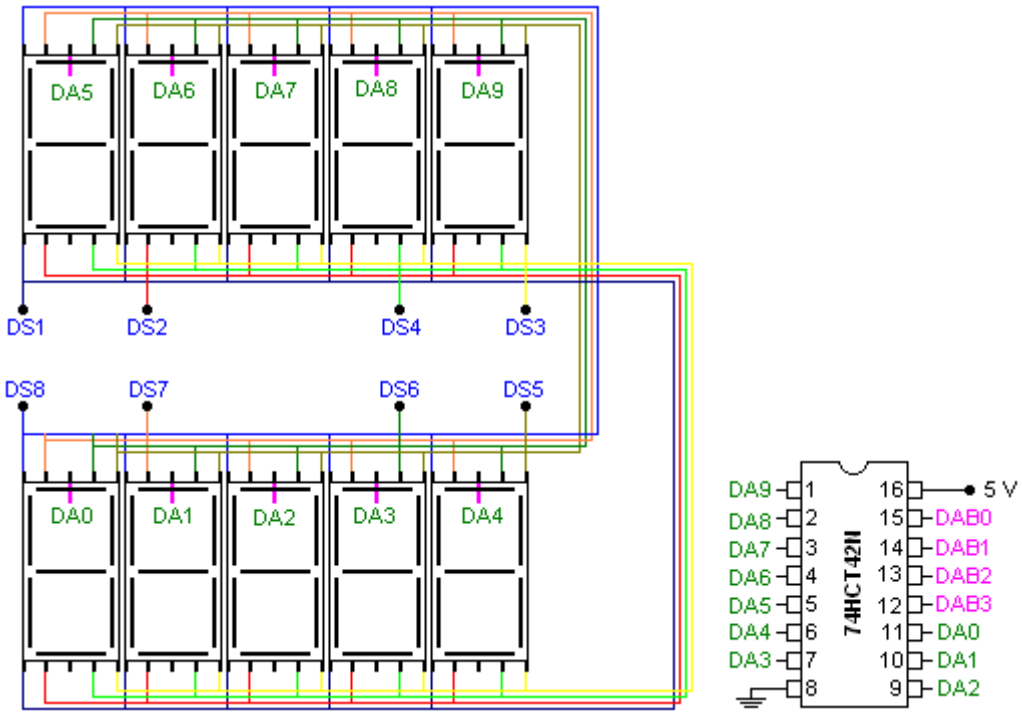


Figura 3.13 – Ligações dos *displays* e do descodificador

3.2.1.2 - Esquema dos Sensores e Interruptores

A interface entre o utilizador e o contador de tempo de bilhar, para além de dois conjuntos de displays, contém também três interruptores de pressão. De modo a saber se o contador se encontra em modo de contagem, em alarme ou em standby, existem quatro sensores colocados na parte traseira da gaveta, que detectarão se as bolas se encontrarem à sua frente. Esses sensores são do tipo microswitch e o esquema de ligação é igual ao esquema de ligação dos interruptores de pressão. Na figura seguinte é apresentado o esquema de ligação para estas duas situações:

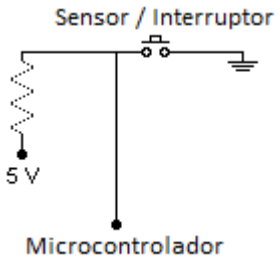


Figura 3.14 – Esquema de ligação dos sensores e dos interruptores

3.2.1.3 - Esquema do Microcontrolador e Unidade Rádio

A comunicação entre a unidade rádio e o microcontrolador, tal como já foi referido é por SPI. O pino SDI de um dispositivo liga ao pino SDO do outro e vice-versa, os pinos SCK, relativos ao clock têm que ser ligados entre si. Relativamente ao módulo de rádio, é também necessário ligar o *chip select,* responsável por seleccionar o dispositivo no caso de haver mais interfaces SPI ligados, é também necessário ligar o *wake* para acordar o dispositivo quando entra em modo *sleep* e é também necessário interligar o sinal de interrupção a um pino de *interrupt-on-change* no microcontrolador. Na figura seguinte, pode-se observar que existem sinais S1 a S4, representando os quatro sensores de detecção de bolas e existem sinais B1 a B3, representando os três interruptores para programação:

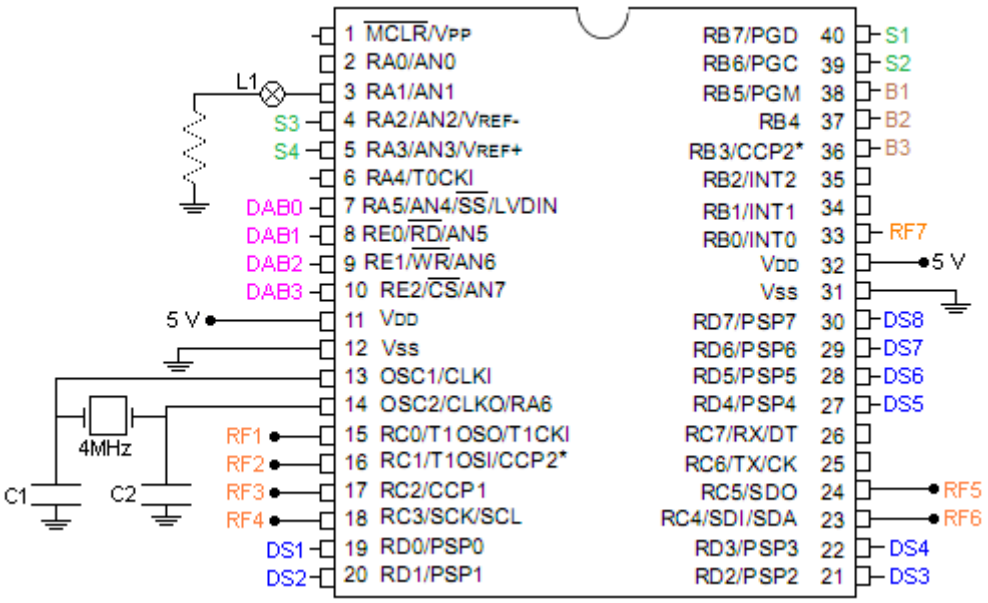


Figura 3.15 – Esquema de ligação do microcontrolador

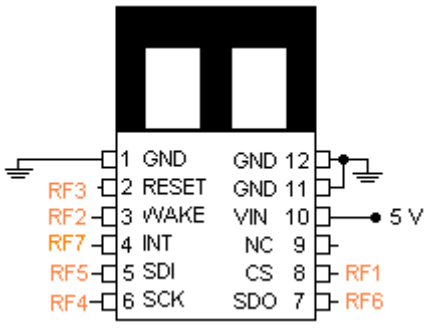


Figura 3.16 – Esquema de ligação da unidade rádio

3.2.1.4 - Desenho do PCB

Após os esquemas de ligação estarem completos, é necessário desenhar o circuito PCB. Existe uma vasta oferta de programas de desenho de PCBs, obrigando a efectuar uma filtragem. Foi dada prioridade a *softwares* gratuitos e o primeiro software a ser experimentado foi o ExpressPCB.

O programa permite desenhar em dupla face, permite a criação de componentes de um modo rápido e é bastante intuitivo. Deste modo, ficou decidido utilizar esta plataforma para o desenho do circuito PCB. Na imagem seguinte pode-se observar o ambiente de trabalho do programa:

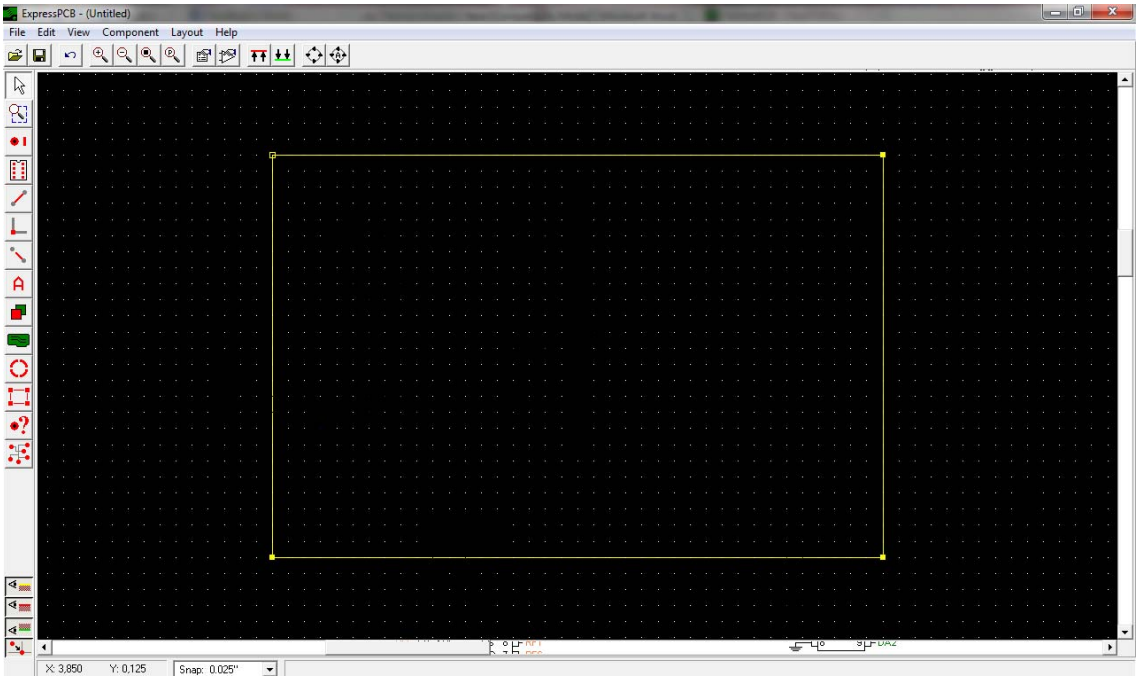


Figura 3.17 – Ambiente de trabalho do ExpressPCB

O circuito PCB desenhado no programa ExpressPCB teve que obedecer a alguns pré-requisitos: localização dos displays e dos interruptores fixas numa determinada posição e tamanho da placa também de um determinado tamanho, para que, caso o projecto passe para produção, possa ser utilizado nos moldes actuais. A placa foi projectada em dupla face e a imagem seguinte apresenta a camada "*top*".

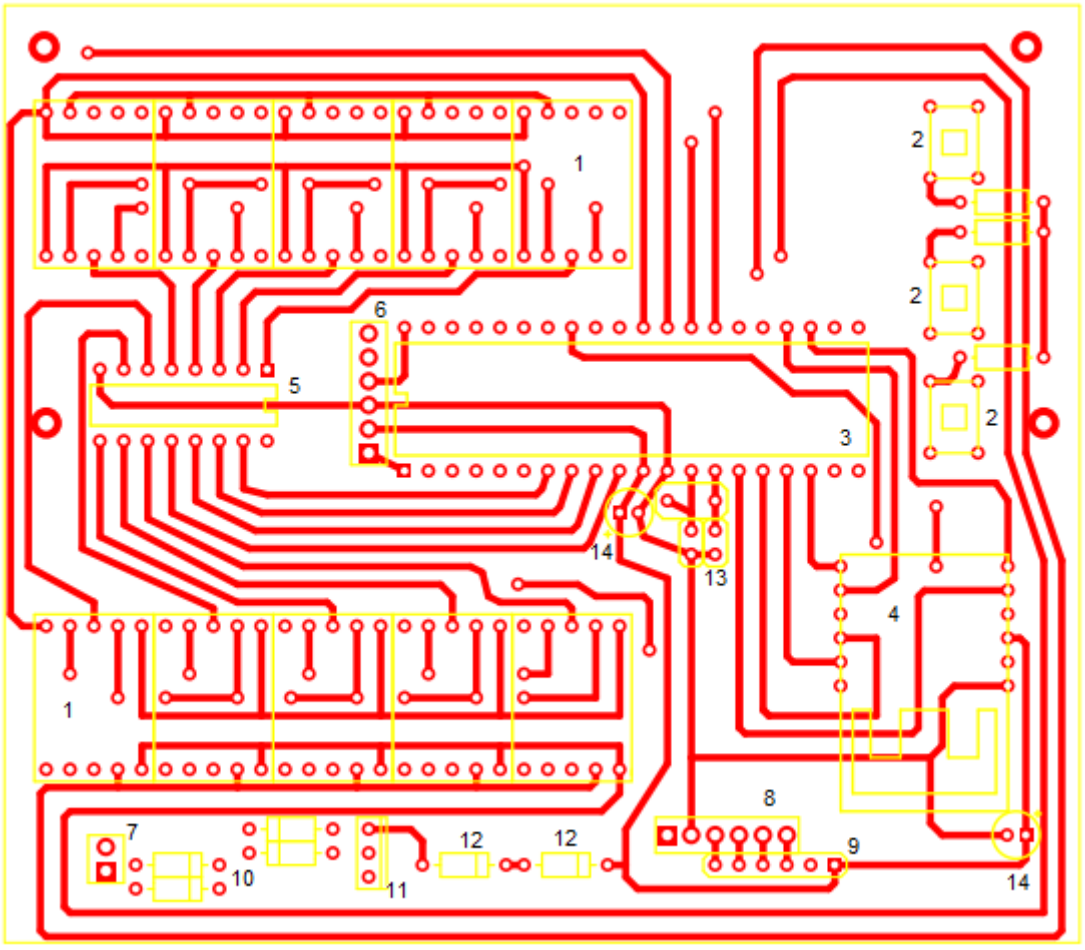


Figura 3.18 - Camada "*top*" do PCB com componentes

Legenda:

1 – Bloco de 5 displays. O superior é para a tarifa e o inferior é para o total a pagar

2 – Interruptores para mostrar o total e programar a tarifa

3 – Microcontrolador Microchip 18F452

4 – Unidade rádio Microchip MRF24J40MA

5 – Descodificador para displays 74HCT42N

6 – Terminal para ligação do programador para ICSP

7 – Terminal para transformador

8 – Terminal para a régua de sensores

9 – Régua de resistências de 10Kohm

10 – Diodos da ponte rectificadora

11 – Regulador de tensão 7805

12 – Diodos para queda de 1,4V de modo a alimentar com segurança a unidade rádio

13 – Conjunto de Cristal de Quartzo de 4MHz e respectivos condensadores

14 – Condensador de desacoplamento

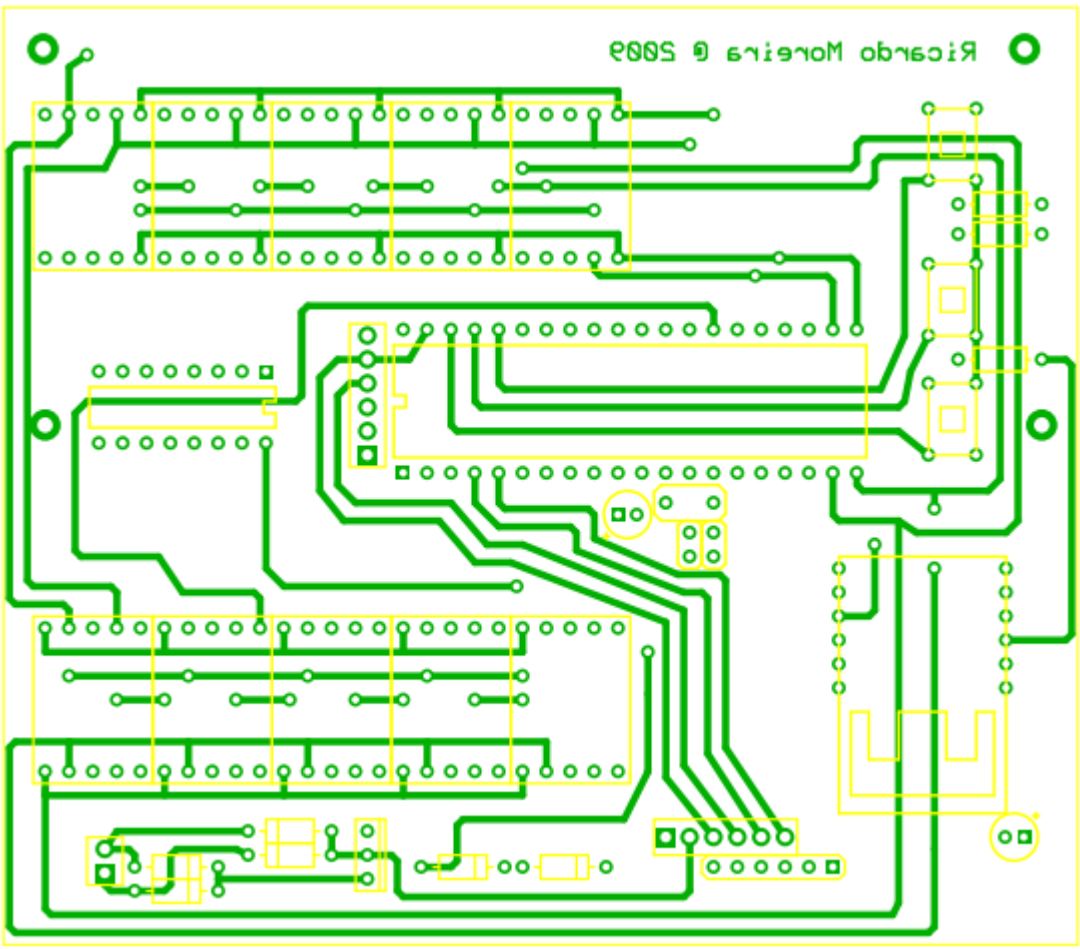


Figura 3.19 – Camada "*bottom*" do PCB com componentes

3.2.1.5 - Produção do Circuito Impresso PCB

Concluído o processo de desenho do circuito impresso, foi procedida a produção dos primeiros circuitos impressos do protótipo. Para tal é necessária uma placa de fibra de vidro revestida a cobre em ambos os lados, equipamento necessário para serigrafia para imprimir o circuito na placa e será também necessário ácido clorídrico e água oxigenada para se dar a corrosão do cobre que está em excesso.

O primeiro passo é o desengorduramento mecânico da placa de cobre. Este processo consiste em esfregar vigorosamente a placa com palha-de-aço, de modo a retirar toda a gordura existente no cobre para que a tinta protectora adira correctamente. O processo pode ser observado na figura seguinte:

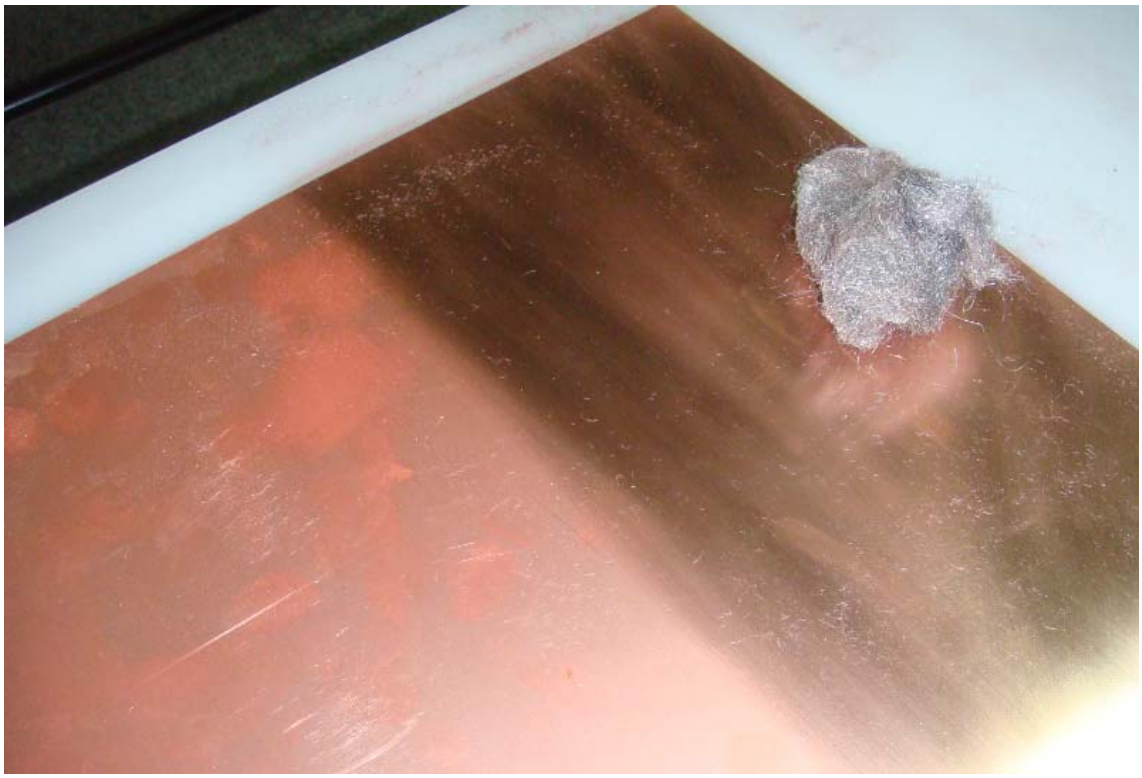


Figura 3.20 – Placa de cobre antes de desengordurar à esquerda e depois à direita

Uma vez concluído o desengorduramento, é necessário efectuar a serigrafia. Para tal, foi necessária a ajuda de uma empresa de espelhos – Signer, que teve a gentileza de abrir um quadro de serigrafia e ajudar nesta parte do projecto. Como a placa a ser utilizada é de dupla face, foi necessário efectuar em primeiro lugar a serigrafia num dos lados da placa. De modo a acertar o posicionamento da placa para a serigrafia do segundo lado, foram efectuados os furos de fixação da placa, e lixada a placa do lado não impresso para que limalhas de cobre não furassem o quadro de serigrafia. Finalmente, com pontos de referência, foi possível efectuar a serigrafia o segundo lado da placa de fibra de vidro revestida a cobre. Na figura seguinte pode ser observado o processo de serigrafia:

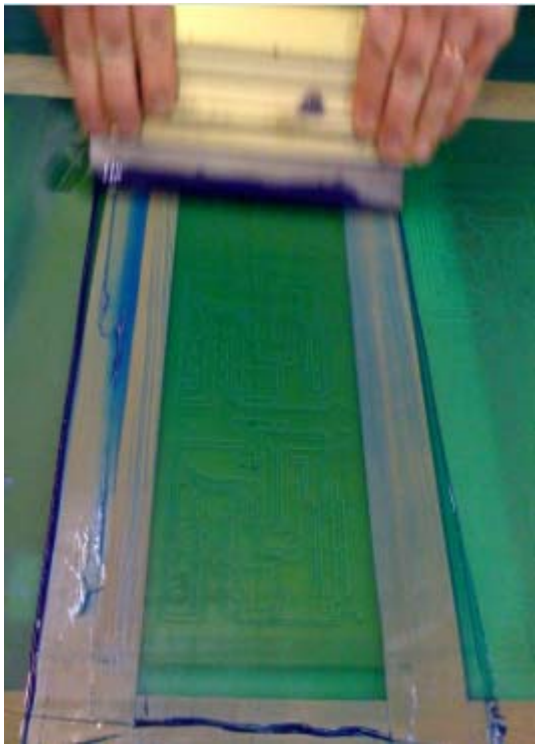


Figura 3.21 – Serigrafia na placa de fibra de vidro revestida a cobre

Uma vez impresso o circuito em ambos os lados da placa, foi mergulhada a placa numa solução ácida para corroer o cobre visível, sobrando apenas o cobre protegido pela tinta, que representa o circuito pretendido. Na figura seguinte pode ser observada a placa com o circuito impresso:

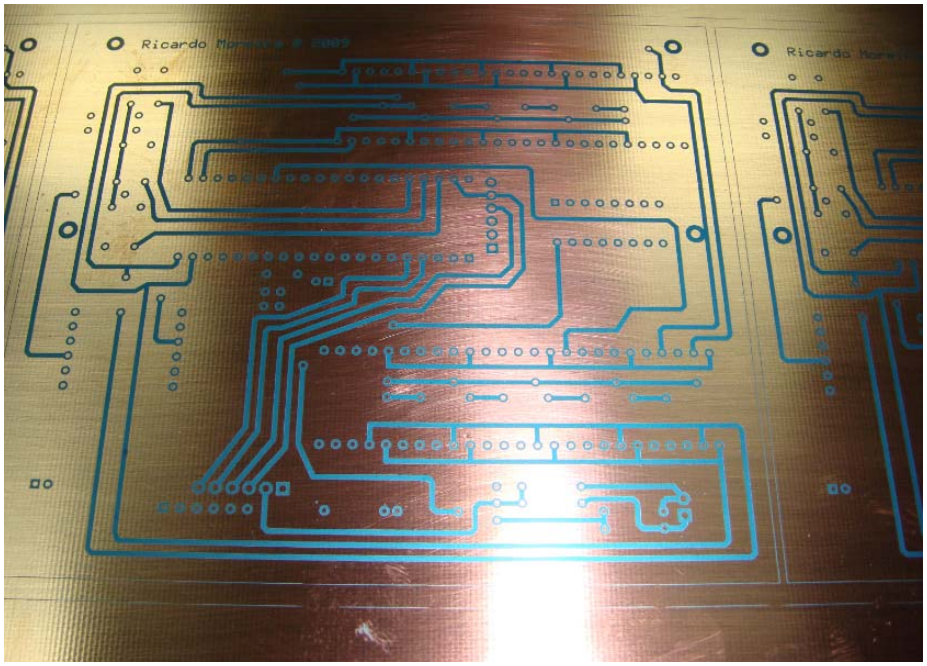


Figura 3.22 – Placa de fibra de vidro revestida a cobre depois da serigrafia

A solução ácida é composta por ácido clorídrico e água oxigenada de 10 volumes, misturados numa proporção aproximadamente de 1:2. O tempo de banho é variável consoante o tamanho da placa e quantidade de pistas, obrigando a uma observação do processo de corrosão cuidada de modo a evitar a corrosão do cobre protegido pela tinta. Neste caso, a corrosão do sobre excessivo demorou aproximadamente 15minutos. Na imagem seguinte pode ser observado o processo de corrosão do cobre:

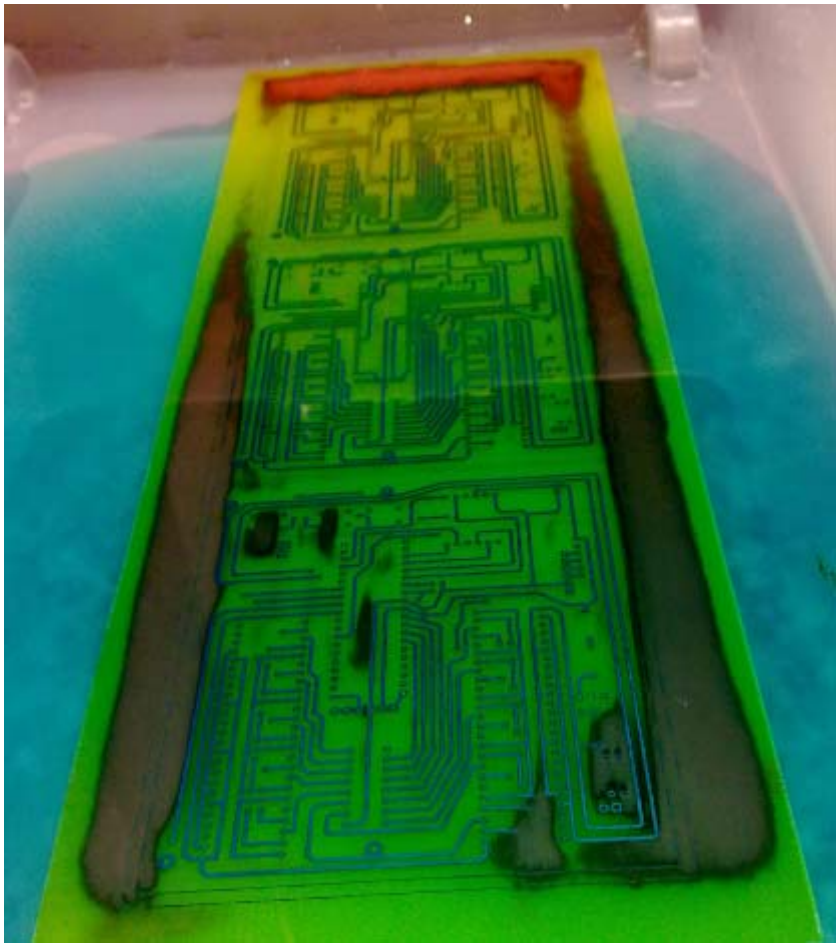


Figura 3.23 – Placa mergulhada na solução ácida

Após o cobre em excesso ter sido corroído, a placa encontrava-se preparada para ser limpa. Para retirar a tinta protectora, existem vários processos, entre os quais esfregar com palha-de-aço, que é um processo moroso, ou mergulhar a placa numa mistura de água com soda cáustica durante uns segundos, que, sendo a solução utilizada, é a solução mais rápida e cómoda.

Um dos processos finais da produção da placa de PCB é a furação para introdução das componentes. Para tal, foi utilizada uma broca adequada à largura de furo pretendida para cada componente. Após a furação concluída, a placa foi esfregada novamente com palha-de-

aço de modo a evitar que as pistas de cobre descolassem da placa de fibra de vidro e para remover as limalhas resultantes da furação.

Finalmente, foi dada uma camada de verniz protector específico para circuitos impressos para evitar a oxidação do cobre e facilitar a soldadura das componentes. O circuito impresso encontrava-se agora pronto para o corte final com o objectivo de separar os três circuitos impressos da placa comum que até ao momento era um só bloco.

Com o circuito impresso completamente terminado, foi possível proceder à colocação das componentes, terminando deste modo o processo de produção e montagem do circuito PCB. Na imagem seguinte pode ser observado o resultado final deste processo:

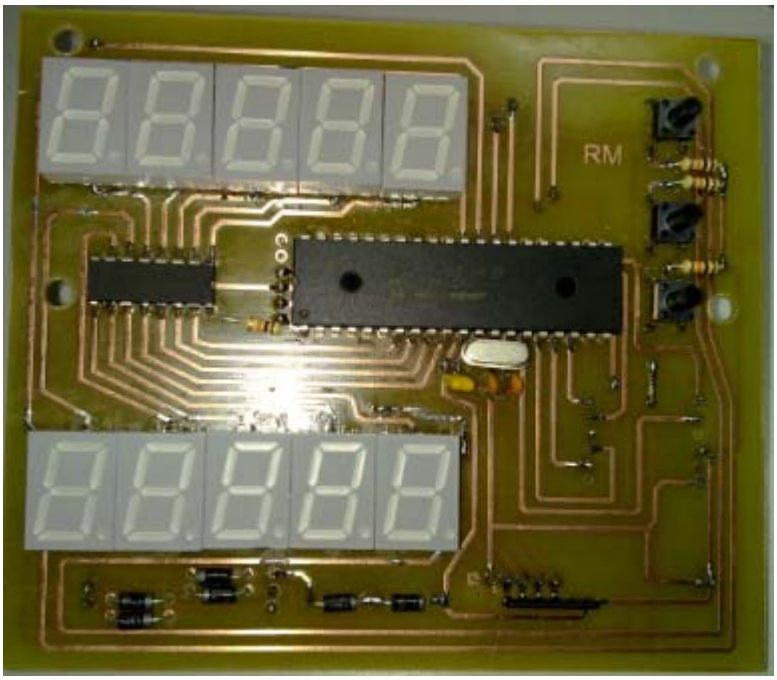


Figura 3.24 – Circuito impresso completamente montado

3.2.2 - Algoritmos e Estruturação de Dados

De acordo com as especificações iniciais, foi verificado que o sistema consiste numa arquitectura distribuída, sendo constituída por um *master* e diversos *slaves*. O master é a unidade responsável por criar a rede sem fios e comunicar com o computador via porta série. Os *slaves* serão os contadores de tempo de bilhar que, através do master, comunicarão com o computador. Desse modo, a estrutura do sistema pode ser ilustrada na figura seguinte:

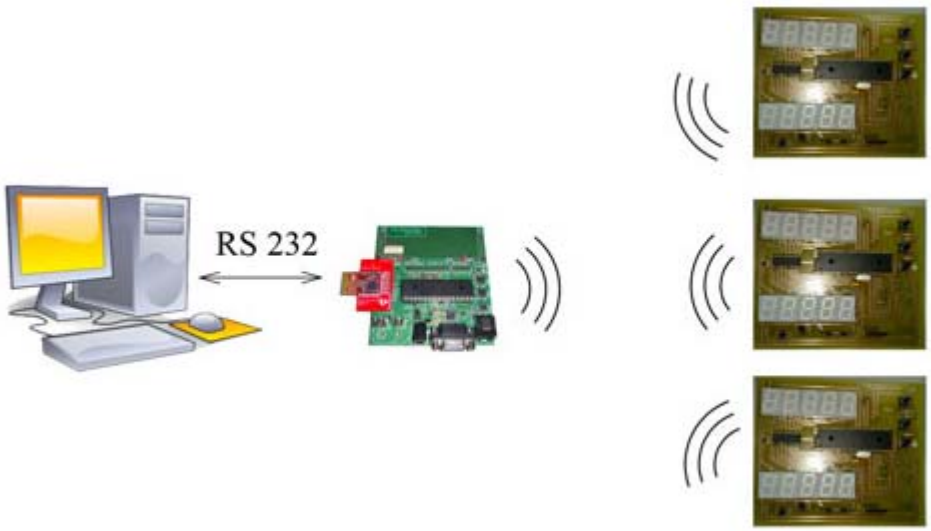


Figura 3.25 – Estrutura do sistema

3.2.2.1 - Placa de Interface (master)

Como se pode observar na imagem anterior, o master é o responsável pela criação e gestão rede sem fios e pela comunicação com o computador.

Uma vez que o kit de desenvolvimento da Microchip contém duas motherboards que comunicam entre si por uma rede sem fios e que comunicam com o computador via protocolo série, foi decidido utilizar uma das motherboards como placa de interface. A nível de hardware a única alteração efectuada foi a colocação de um terminal para efectuar a programação on-board e evitar deste modo a remoção do microcontrolador sempre que houvesse uma alteração ao programa.

Figura 3.26 – Interface de comunicações

Funcionalmente, a placa de interface (master) irá funcionar como um *buffer*, em que irá transferir a informação que recebe por rede sem fios para o computador através da rede cablada e vice-versa. Como tal, o diagrama de blocos é bastante simples e pode ser observado na seguinte imagem:

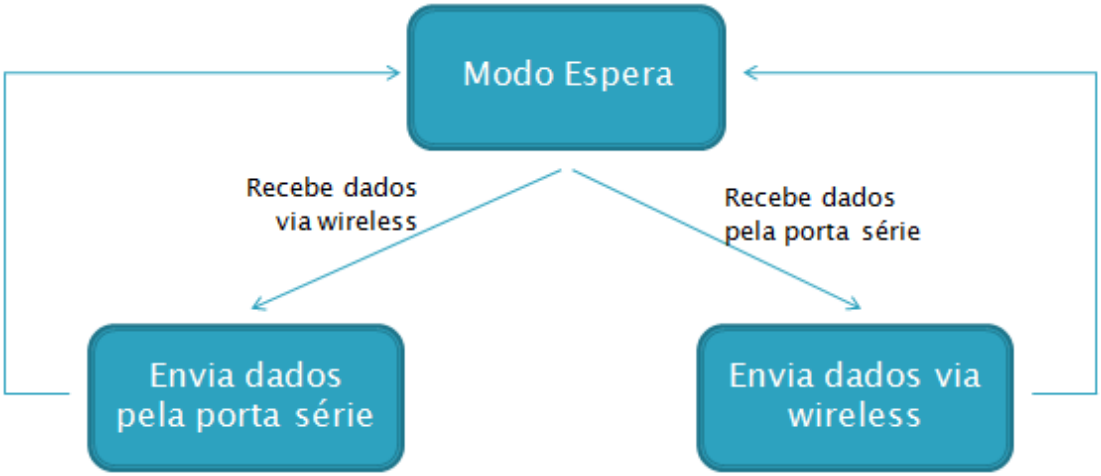


Figura 3.27 – Diagrama de blocos da placa de interface

A programação do microcontrolador, tal como já foi referido, é em C, utilizando o MPLAB da Microchip e o compilador MCC C18. Analisando o diagrama de blocos, verifica-se que o código é relativamente simples:

- Se a interface de comunicações recebe dados vindos de um contador de tempo de bilhar, esta adiciona o caracter "%" ao fim da trama de modo a facilitar a interpretação dos dados na aplicação em Visual Basic e envia pela porta série.
- Se a interface de comunicações recebe dados vindos do computador, esta efectua um *broadcast* da informação recebida.

O seguinte código refere-se ao reenvio dos dados recebidos por *wireless* para a porta série. O bit *ReceivedPacket*() é automaticamente activado pela MiWi P2P *Stack* sempre que um *packet* é recebido, o comando ConsolePut() refere-se à função de envio de uma *string* pela porta série enquanto que a função DiscardPacket() tem como objectivo limpar o buffer e reiniciar o Rx do módulo MRF24J40.

```
if( ReceivedPacket() )
{
    for(i = 0; i < rxFrame.PayLoadSize; i++)
    {
        ConsolePut(rxFrame.PayLoad[i]);
    }
DiscardPacket();
}
```

Código 3.1 – Reencaminhamento da trama via RS232

Para receber dados da porta série, é necessário ler os seus valores periodicamente. O comando ConsoleIsGetReady(), tal como o nome indica, é activo sempre que existam dados prontos a serem lidos na porta série. O comando Flushtx() efectua um reset ao buffer de transmissão sem fios. Para

enviar dados via *wireless*, existem dois métodos: UniCast e BroadCast. O primeiro tem como objectivo enviar para um só dispositivo, sendo que o segundo tem como objectivo enviar para todos os dispositivos adjacentes ao coordenador. Foi decidido utilizar o método de BroadCast e o próprio contador de tempo de bilhar irá fazer a filtragem dos dados, respondendo apenas caso seja o destinatário.

```
if ( ConsoleIsGetReady())
{
    BYTE x=46;
    ConsoleGetString(var_recebida,x);
    FlushTx();
    for(i = 0; i < 46; i++)
    {
        WriteData(var_recebida[i]);
    }
    BroadcastPacket(myPANID, FALSE, FALSE);
}
```

Código 3.2 – Reencaminhamento da trama via *wireless*

3.2.2.2 - Placa do Contador de Tempo de Bilhar (slave)

Observando a figura 3.26, pode-se concluir que a placa do contador de tempo de bilhar é o *slave* da comunicação sem fios.

O modo de funcionamento pode ser demonstrado pelo seguinte diagrama de blocos:

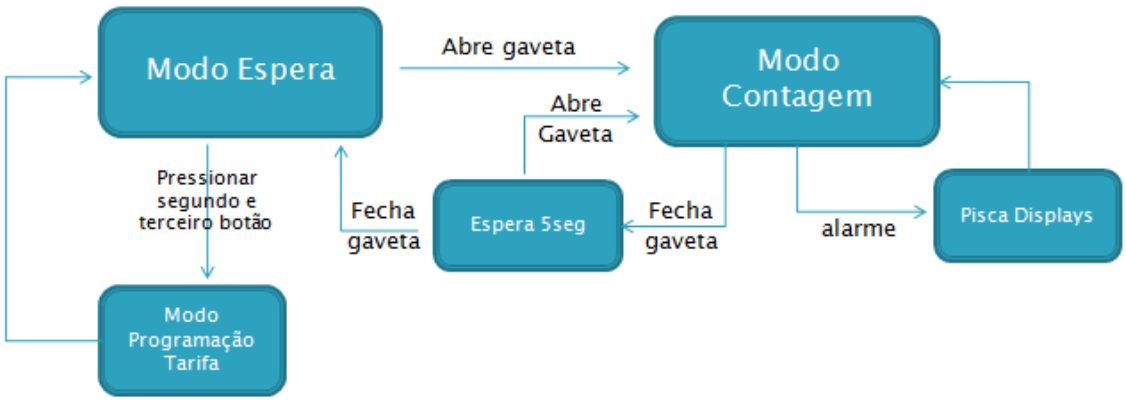


Figura 3.28 – Diagrama de blocos do contador de tempo de bilhar

Tal como acontece com a placa de interface, o microcontrolador do contador de tempo de bilhar é programado em C, utilizando a plataforma MPLAB da Microchip e o compilador MCC C18.

Comparativamente com a placa de interface, o código do programa do contador de tempo de bilhar é mais complexo pois existe interface com displays, sensores, interruptores e efectuar uma contagem precisa do tempo.

Sabendo que as comunicações sem fio utilizam o timer0 para retorno do tempo, foi decidido utilizar o timer2 para o contador de fracções relativo aos impulsos de contagem, com um tempo de interrupção de 16,384ms. O timer3 foi utilizado para todas as outras operações que obriguem a tempos de espera e foi decidido utilizar o tempo de interrupção de 131,072ms.

O contador de tempo de bilhar inicia-se no modo "normal". Ao entrar no ciclo infinito, caso o contador esteja no modo normal, ou seja, de contagem, irá executar a função verifica_sensores().

Após concluir, verifica se recebeu alguma trama via *wireless*. Caso positivo, o microcontrolador verifica se os primeiros dois bits se referem à leitura ou á escrita. Caso seja de leitura, o microcontrolador efectua uma leitura aos valores da EEPROM e envia-os via *wireless* para a placa de interface. Caso os dois primeiros bits sejam de recepção, o microcontrolador substitui os valores da EEPROM pelos valores recebidos.

Relativamente às interrupções, existem 2 timers sempre em funcionamento: timer2 e timer3. O timer2 gera uma interrupção a cada 16,384ms. Consoante a tarifa e fracção de contagem, o número de ciclos que têm que ser efectuados de modo a obter o tempo para incrementar o total a pagar é calculado e gravado na variável n_ciclos_contagem_f. Sempre que existe uma interrupção, o número do ciclo actual é comparado com o número de ciclos que deve efectuar, e caso seja igual ou superior, o total a pagar é incrementado da fracção, este valor é adicionado ao total facturado e gravado na EEPROM e caso esteja em modo rede, este envia uma trama para a placa de interface para informar que houve mudança de valor a pagar.

O timer3 tem como objectivo esperar cinco segundos (38 ciclos) após a gaveta ser fechada e determinar se o estado é o mesmo, de modo a evitar fechos de gaveta ou resets não pretendidos; piscar os displays em caso de alarme de fecho (3 ciclos) de gaveta sem todas as bolas; e finalmente piscar o display activo em caso de programação (1 ciclo).

Para gravar e ler valores na EEPROM, é necessário cumprir com os procedimentos descritos na *datasheet*. De modo a não sobrecarregar o código, foram criadas duas funções e sempre que houvesse necessidade de ler ou escrever efectuava-se o chamamento do código.

```
void Grava_EEP(unsigned int eep_endereco, unsigned int eep_valor)
{
    EEADR =eep_endereco;
    EEDATA = eep_valor;
    EECON1bits.EEPGD =0;
    EECON1bits.CFGS =0;
    EECON1bits.WREN =1;
    INTCONbits.GIE = 0;
    EECON2 = 0x55;
    EECON2 = 0xAA;
    EECON1bits.WR = 1;
    INTCONbits.GIE = 1;
    EECON1bits.WREN = 0;
    Delay100TCYx(155);
    Delay100TCYx(155);
    Delay100TCYx(155);
}
```

Código 3.3 – Gravação de valores na EEPROM

```
unsigned int LER_EEP(unsigned int eep_endereco)
{
unsigned int valor_lido;
EEADR =eep_endereco;
EECON1bits.EEPGD = 0;
EECON1bits.CFGS =0;
EECON1bits.RD = 1;
valor_lido=EEDATA;
Delay100TCYx(155);
return valor_lido;
}
```

Código 3.4 – Leitura de valores da EEPROM

A função verifica_sensores(), tal como foi referido acima, é a primeira função do ciclo *while*, caso o contador esteja no modo normal. Esta função executa vários processos:

Caso os sensores estejam abertos e o estado anterior da gaveta era:

- "fechado", reinicia o valor da fracção, define o estado da gaveta para "em_jogo" e mostra a tarifa e o total a pagar. Caso esteja no modo rede, enviará uma trama a avisar que o jogo foi aberto.
- "em_jogo", apenas mostra a tarifa e o total a pagar
- "alarme", define o estado da gaveta para "em_jogo" e mostra a tarifa e o total a pagar
- "just_closed", define o estado da gaveta para "em_jogo" e mostra a tarifa e o total a pagar

Caso os sensores estejam em erro e o estado anterior da gaveta era:

- "em_jogo", define o estado da gaveta para "alarme", mostra a tarifa e o total a pagar e define o bit estado_displays com o valor zero
- "alarme" e caso o bit estado_displays estiver activo, mostrará a tarifa e o total a pagar

Caso os sensores estejam fechados e o estado anterior da gaveta era:

- "fechada" e caso os interruptores 2 e 3 forem pressionados simultaneamente, irá para o modo de programação de tarifa
- "em_jogo", define o estado da gaveta para "just_closed" e mostra a tarifa e o total a pagar
- "alarme", define o estado da gaveta para "just_closed" e mostra a tarifa e o total a pagar
- "just_closed", mostra a tarifa e o total a pagar

A programação da tarifa é iniciada, tal como já foi referido, quando a gaveta está fechada e os interruptores 2 e 3 são pressionados ao mesmo tempo. Após isso, o primeiro display irá piscar e através do segundo e terceiro interruptor (incrementar e cursor respectivamente), insere-se a *password* e posteriormente pressiona-se o primeiro interruptor (confirmação). Caso a *password* esteja correcta, a tarifa irá aparecer nos displays superiores e através do segundo e terceiro interruptor poder-se-á definir a nova tarifa e novamente com o interruptor superior confirmar-se-á. Caso a *password* esteja incorrecta, o modo de programação de tarifa será abandonado.

As tramas enviadas pelo contador de tempo de bilhar são:

```
$#SN0X5X1X264#$          , para o modo rede
$#SN0X5X1XF3XPD31XT01600XTF950994XPW12345#$%    , para o modo leitura individual
```

Código 3.5 – Leitura de valores da EEPROM

Explicação das tramas:

Código que inicia a trama: "$#"

Código que finaliza a trama: "#$"

O valor "X" indica que o valor da variável terminou e que a partir desse momento será outra variável.

SN -> *Serial Number*

F -> Fracção

PD -> *Power Down*

T01 -> Tarifa

TF -> Total Facturado

PW -> *Password*

O *serial number* é composto por três conjuntos de dados, sendo que o primeiro representa o ano, o segundo representa o número da semana e o último o número de produção.

No caso do modo rede, após o número de série, o valor seguinte representa o total a pagar. Caso em vez de um número a trama contiver "FG", essa trama indica que houve fecho de gaveta.

3.2.2.3 - Desenvolvimento da Aplicação em Visual Basic

De modo a ser possível efectuar a monitorização e controlo da rede, é necessário o desenvolvimento de uma aplicação para executar num computador. A aplicação foi desenvolvida em Visual Basic e o seu funcionamento tem dois objectivos bem distintos: ler e alterar os dados armazenados na eeprom de um contador de tempo de bilhar (só pode ser o único ligado à rede uma vez que se configura, entre outros, o serial number) e servir de interface gráfica, permitindo que seja facilmente visível o modo em que se encontra cada contador de tempo de bilhar.

Foi decidido dividir o menu da aplicação em cinco *tabs*: "Modo Rede", "Leitura Individual", "Configuração de Rede", "Comunicação" e "Consultas".

No primeiro *tab*, encontram-se dispostas dez mesas de bilhar, em que cada uma representa um contador de tempo. Caso o salão tenha apenas dois bilhares, os bilhares número três a número dez, encontrar-se-ão a preto e branco. Os bilhares activos estarão a cores. Caso a mesa de bilhar se encontre em jogo, a imagem da mesa de bilhar estará também em jogo com as bolas em cima e o valor a pagar estará a ser mostrado. Sempre que o jogo é fechado, a imagem mudará para um bilhar sem bolas e aparecerá uma *label* a anunciar pagamento e com o respectivo valor a mostrar.

A *tab* "Leitura Individual" consistirá em ler e/ou gravar os valores da EEPROM do contador de tempo de bilhar, nomeadamente o número de série, a fracção mínima, o número de *power*

downs, a tarifa, o total facturado e a *password* para configuração da tarifa. O *tab* seguinte servirá apenas para definir os números de série atribuídos a cada mesa de bilhar. O *tab* "Comunicação" será um tab que na versão final ficará oculta pois consiste em registar as tramas enviadas ou recebidas e permitirá o envio manual de tramas. Finalmente, a última *tab*, consiste em efectuar a consulta do histórico, quer a nível de total acumulado, quer a nível de número de partidas de cada contador de tempo de bilhar.

Figura 3.29 – Screenshot da aplicação em Visual Basic no modo rede

Ao iniciar a *form*, é necessário configurar a porta série para permitir a recepção dos dados:

```
Private Sub Form_Load()
    ' Fire Rx Event Every Two Bytes
    MSComm1.RThreshold = 1
    ' When Inputting Data, Input 2 Bytes at a time
    MSComm1.InputLen = 1
    ' 2400 Baud, No Parity, 8 Data Bits, 1 Stop Bit
    MSComm1.Settings = "19200,N,8,1"
    ' Disable DTR
    MSComm1.DTREnable = False
    ' Open COM1
    MSComm1.CommPort = 1
    MSComm1.PortOpen = True
End Sub
```

Código 3.6 – Código a executar ao iniciar a *form*

Se houver recepção de dados, estes serão enviados para uma *textbox* que representará o histórico e a partir daí será feita a análise da trama. Uma vez que nas tramas utilizadas não se utilizou o caracter "%", a placa de interface adiciona este caracter ao fim de cada trama enviada pelos contadores de tempo de bilhar para que o Visual Basic interprete o fim de cada trama.

```
Private Sub MSComm1_OnComm()
    Dim sData As String ' Holds our incoming data
    ' If comEvReceive Event then get data and display
    If MSComm1.CommEvent = comEvReceive Then
       sData = MSComm1.Input ' Get data (1 byte)
       ' Convert value to a string and display
       Text1.Text = Text1.Text & sData
       If sData = "%" Then
          txt_recebido.Text = trama_recebida
          trama_recebida = ""
       Else
          trama_recebida = trama_recebida & sData
       End If
    End If
End Sub
```

Código 3.7 – Código a executar ao receber uma trama

Cada trama que o Visual Basic envie, terá que conter 46 caracteres pois são os necessários para a comunicação. Existem quatro tipos de tramas que o Visual Basic envia: trama de pedido de leitura de valores de EEPROM, trama de escrita de valores de EEPROM, trama de activação do modo de rede num determinado contador de tempo de bilhar e trama de activação do modo normal num determinado contador de tempo de bilhar.

```
    Pedido de Leitura:
$%@@@*
    Gravação de Dados:
$#00900500100302023400010440001234560000000000#$
    Definição do modo normal
!R009005001@@@*
    Definição do modo de rede
!N009005001@@@*
```

Código 3.8 – Tramas enviadas pelo Visual Basic

Explicação das tramas:

A primeira trama, como referido acima, refere-se ao pedido de leitura dos valores da EEPROM ao contador de tempo de bilhar. Os caracteres "$%" indicam que está a pedir leitura.

* Os caracteres @ são caracteres sem significado e têm como objectivo completar a trama até perfazer os 46 caracteres.

A trama de gravação de dados consiste em enviar os valores que se pretendem gravar no contador de tempo de bilhar, e são enviados em pacotes de três dígitos, cada bloco entre 0 e 255, perfazendo os oito bits de cada posição de memória.

Substituindo os números por variáveis:

S# SNA SNS SNN FR PDH PDL TH TL TFH TFM TFL PWH PWM PWL #$, sendo que:

SNA -> ano do número de série

SNS -> número de semana do número de série

SNN -> número de produção do número de série

FR -> fracção

PDH -> 8 bits mais significativos do *powerdown*

PDL -> 8 bits menos significativos do *powerdown*

TH -> 8 bits mais significativos da tarifa

TL -> 8 bits menos significativos da tarifa

TFH -> 8 bits mais significativos do total facturado

TFM -> 8 bits intermédios do total facturado

TFL -> 8 bits menos significativos do total facturado

PWH -> 8 bits mais significativos da *password*

PWM -> 8 bits intermédios da *password*

PWL -> 8 bits menos significativos do *password*

Relativamente á definição do modo normal e modo de rede, os nove dígitos referem-se ao número de série ao qual a mensagem é destinada. Os caracteres no início "!R" refere-se á activação do modo de rede e os caracteres "!N" referem-se à activação do modo normal.

Neste capítulo foi desenvolvido o sistema nas suas componentes hardware e software e validado o seu funcionamento. O seu comportamento face aos objectivos iniciais vai ser analisado no capítulo seguinte com análise de resultados e algumas conclusões sobre os mesmos.

Capítulo 4

Resultados e Análise

Consoante as especificações iniciais, a dissertação consiste no estudo, projecto, desenvolvimento e montagem de um ou mais protótipos de um contador de tempo de bilhar que funcionasse autonomamente, ou que, através de uma placa de interface, houvesse monitorização e controlo de toda a rede no computador. Todos os requisitos com excepção de um foram obtidos em pleno. O único requisito que não foi cumprido em pleno deve-se ao facto de o tempo de transmissão e processamento de dados serem superiores ao tempo de varrimento dos displays, tendo como consequência um apagão dos *displays* no instante em que recebe ou envia uma trama via *wireless*. Este apagão tem duração igual ao tempo de transmissão e processamento da trama recebida ou enviada.

O projecto foi desenvolvido para compreender a interligação de um sistema com 128 contadores de tempo de bilhar à placa de interface, no entanto o protótipo tem apenas duas placas de contador de tempo de bilhar uma vez que apenas se disponibilizaram três unidades rádio em tempo útil.

A aplicação em Visual Basic foi desenvolvida de modo a poder controlar até 10 contadores de tempo, valor suficiente tendo em conta o número de bilhares por salão de jogos.

Os ensaios realizados permitiram testar o alcance e foi medido um alcance de cerca de 15metros num ambiente com paredes, resultados aquém do esperado e anunciado pelo fabricante.

Este problema de alcance reduzido é geral, o que obrigou a Microchip a desenvolver um novo módulo de unidade rádio, baseado no mesmo controlador, com um amplificador de antena melhorado, anunciando um alcance dez vezes superior ao anunciado com o módulo utilizado no desenvolvimento deste projecto.

Uma última característica a analisar nos resultados é o erro máximo admissível, uma vez que o de acordo com a legislação em vigor do Instituto Português da Qualidade, o erro máximo admissível para este tipo de equipamentos é de 1%.

O cálculo do erro de contagem pode ser conferido seguindo as fórmulas seguintes:

Tempo necessário para aumentar a fracção:

$$t1(s)= ((\text{fracção}*60) / \text{tarifa}) * 60 \qquad (1)$$

Número de interrupções necessárias para aumentar a fracção:

$$n1= (t1 * 1000) / 16{,}384 \qquad (2)$$

Número de interrupções tem que ser um valor unitário, portanto apenas a parte inteira do valor anterior obtido será considerado, sendo que n'1=int(n1)

Tempo real de aumento de fracção será:

$$t2(s) = (n'1 * 16{,}384) / 1000 \qquad (3)$$

Finalmente, o erro será calculado pela expressão:

$$e=(t1 - t2) / t1 \qquad (4)$$

Como se pode confirmar pelas fórmulas anteriores, o erro de contagem varia consoante a relação entre a fracção e a tarifa. Em Portugal, a fracção utilizada é de 5 cêntimos, portanto a tabela seguinte apresenta uns valores de tarifa comuns e verificação do erro:

Tabela 4.1 — Tabela de exemplos de cálculo do erro de contagem

Fracção (cêntimos)	Tarifa (cêntimos)	Erro de Contagem (%)
5	250	0,0121
5	300	0,0030
5	450	0,0166
5	500	0,0121
5	600	0,0030
5	700	0,0303
5	800	0,0212
5	900	0,0576
5	1000	0,0576

Como se pode observar na tabela anterior, o erro máximo do sistema é de 0,0576%, valor muito inferior ao 1% admitido (aproximadamente 17 vezes inferior).

Capítulo 5

Conclusões

Para a realização desta dissertação, foi decidido efectuar um projecto que aperfeiçoasse o sistema existente relativo à interligação de contadores de tempo de bilhar em rede. O projecto existente é baseado em ligação cablada, utilizando o protocolo RS485.

De modo a completar o sistema já disponível no mercado, foi decidido utilizar uma tecnologia sem fio que suportasse o modo rede, substituindo as convencionais ligações por cabo, vantajosa quer a nível de eliminação de cablagem, mas também da flexibilidade das mesas de bilhar.

Do ponto de vista de controlo, foi decidido utilizar o mesmo microcontrolador (Microchip 18F452) pois contém todas as características necessárias para a sua implementação e por reduzir os custos de armazenamento. O módulo de unidade rádio utilizado no desenvolvimento deste projecto foi o Microchip MRF24J40MA. No entanto, os resultados deste trabalho mostram que será preciso trabalhar com o módulo MRF24J40MB, que graças ao seu amplificador de sinal anuncia um alcance de dez vezes superior. Não foi possível utilizar este módulo desde o início do projecto uma vez que o lançamento sofreu um atraso por parte da Microchip e já só foi lançado em meados do desenvolvimento do projecto.

O objectivo que tinha sido proposto atingir nesta dissertação, foi realizar o desenvolvimento de um contador de tempo de bilhar que comunicasse com uma aplicação a ser executada num computador através de uma tecnologia sem fios. Este objectivo foi alcançado, demonstrando-se assim que as tecnologias baseadas na especificação IEEE 802.15.4 tem potencialidades que permitem melhorar o desempenho de sistemas distribuídos de automação baseados tem tecnologias convencionais.

O projecto desenvolvido no âmbito desta dissertação é, como inicialmente foi referido, bastante limitado, tendo como principal vantagem a sua versatilidade por não necessitar de efectuar uma cablagem dedicada. Certamente não será um projecto com tempo de vida curto uma vez que as tecnologias sem fios estão cada vez mais presentes no nosso dia-a-dia.

Referências

[1] Dados baseados em http://en.wikipedia.org/wiki/Field-programmable_gate_array, último acesso em 22 de Março de 2010

[2] Imagem retirada de http://es.wikipedia.org/wiki/FPGA#Programaci.C3.B3n, último acesso em 22 de Março de 2010

[3] Dados baseados em http://www.i-magazine.com.br/imagazine/picbook/cap1.htm, último acesso em 14 de Março de 2010

[4] Dados baseados em http://encyclopedia.thefreedictionary.com/fpga, último acesso em 22 de Março de 2010

[5] Dados baseado em http://en.wikipedia.org/wiki/ZigBee#History, último acesso em 22 de Março de 2010

[6] Dados e imagem baseados em http://www.projetoderedes.com.br/artigos/artigo_zigbee.php, último acesso em 24 de Fevereiro de 2010

[7] Dados baseados na "Microchip Stack for the ZigBeeTM Protocol", Disponível em http://ww1.microchip.com/downloads/en/AppNotes/00965c.pdf, , último acesso em 2 de Fevereiro de 2010

[8] Texto baseado em http://en.wikipedia.org/wiki/ZigBee_specification, último acesso em 24 de Fevereiro de 2010

[9] Dados baseados na "Microchip MiWiTM Wireless Networking Protocol Stack". Disponível em http://ww1.microchip.com/downloads/en/AppNotes/MiWi%20Application%20Note_AN1066.pdf, último acesso em 22 de Março de 2010

[10] Tamanho de código obtido através da compilação dos exemplos fornecidos pela Microchip

[11] Dados baseados na "Microchip MiWiTM P2P Wireless Protocol". Disponível em http://ww1.microchip.com/downloads/en/AppNotes/AN1204%20-%20MiWi%20P2P %20App %20Note.pdf, último acesso em 22 de Março de 2010

[12] Dados baseados na definição de FPGA da Wikipédia. Disponível em http://en.wikipedia.org/wiki/Field-programmable_gate_array, último acesso em 10 de Dezembro de 2009

[13] Dados baseados na definição de Microcontrolador da Wikipédia. Disponível em http://en.wikipedia.org/wiki/Microcontroller, último acesso em 10 de Dezembro de 2010

[14] Dados baseados no website da ZigBee Alliance. Disponível em http://www.zigbee.org, último acesso em 22 de Março de 2010

[15] Dados baseados no site "Palo Wireless ZigBee Resource Center". Disponível em http://www.palowireless.com/zigbee , último acesso em 12 de Dezembro de 2009

[16] Dados baseados no site da Digi ZigBee. Disponível em http://www.digi.com/technology/rf-articles/wireless-zigbee.jsp, último acesso em 12 de Dezembro de 2009

[17] Dados baseados na datasheet da unidade rádio Microchip MRF24J40. Disponível em http://ww1.microchip.com/downloads/en/DeviceDoc/DS-39776b.pdf, último acesso em 18 de Dezembro de 2009

[18] Dados baseados na datasheet do módulo de unidade rádio Microchip MRF24J40MA. Disponível em http://ww1.microchip.com/downloads/en/DeviceDoc/70329b.pdf, último acesso em 18 de Dezembro de 2009

[19] Dados e Imagens baseadas na datasheet do microcontrolador Microchip 18F452. Disponível em http://ww1.microchip.com/downloads/en/DeviceDoc/39564c.pdf, último acesso em 20 de Dezembro de 2009

[20] Dados baseados na datasheet do driver de displays 74HCT42N. Disponível em http://www.datasheetcatalog.org/datasheet2/5/0qgptkedu06gqg3d377h7cc55e3y.pdf, último acesso em 22 de Dezembro de 2009

Anexos

a) Código em C

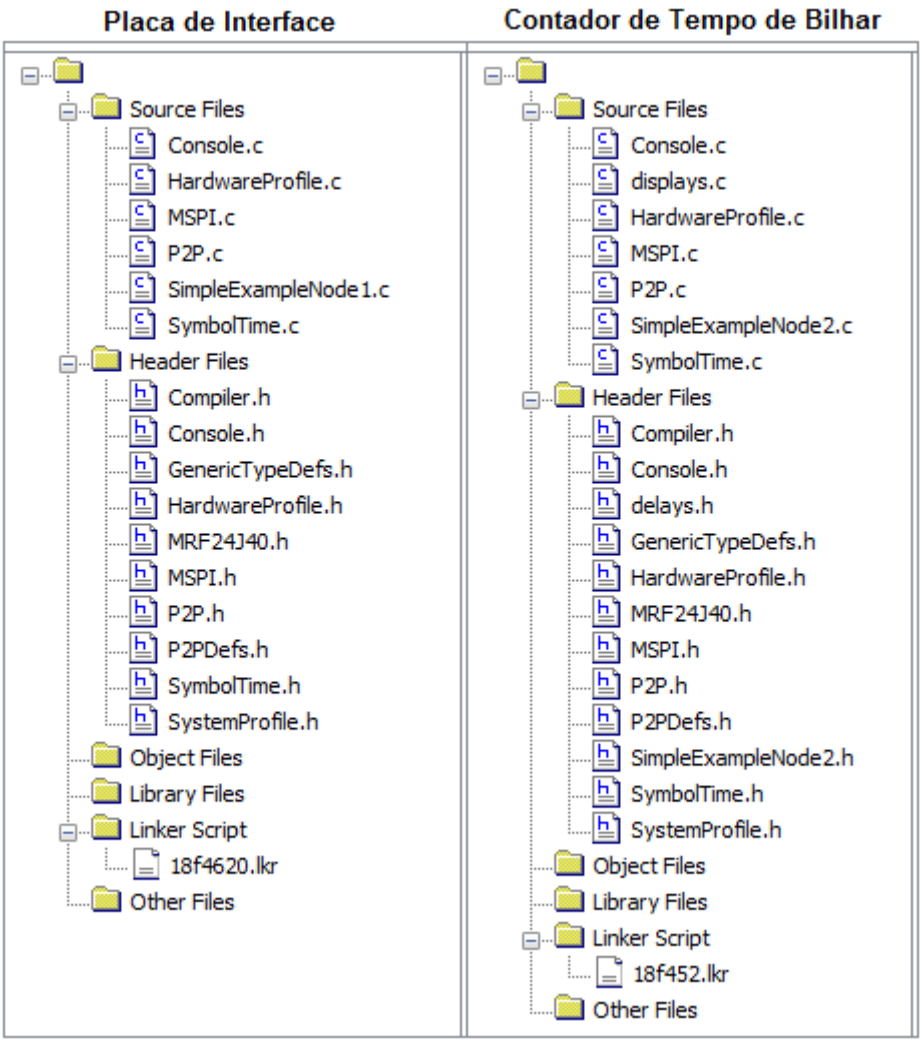


Figura 7.1 – Ficheiros incluídos em cada projecto C

A figura anterior apresenta os ficheiros incluídos em cada projecto. O projecto da placa de interface contém os ficheiros na lista da esquerda e o projecto do contador de tempo de bilhar contém os ficheiros na lista da direita.

O ficheiro “console.c” representa o bloco de código responsável pela comunicação série: configuração, envio e recepção e é comum a ambos os projectos.

O ficheiro “HardwareProfile.c” é responsável pela configuração do hardware: configurar portos de entrada e saída, interrupções e SPI. Existem duas versões, uma para a placa de interface e outra para os contadores de tempo de bilhar.

O código de envio e recepção de bytes através da SPI encontra-se no ficheiro "MSPI.c".

Todas as funções relativas ao Peer-To-Peer da tecnologia MiWi encontra-se no ficheiro "P2P.c". Este ficheiro corresponde ao protocolo de comunicações.

O código a executar pela placa de interface encontra-se no ficheiro "SimpleExampleNode1.c". Este é o código responsável por fazer o reencaminhamento de dados.

O ficheiro "SimpleExampleNode2.c" é o responsável pelo código principal do contador de tempo de bilhar: configura os timers, cria tramas para envio, interpreta tramas recebidas, lê e grava valores na EEPROM, faz a contagem do tempo de jogo, verifica o estado dos sensores, etc.

O ficheiro "SymbolTime.c" contém o código responsável por configurar a UART e por lêr o valor do tempo para evitar *rollover* de dados na leitura e/ou envio de tramas.

O código associado aos displays está incluído no ficheiro "Displays.c". Os displays estão multiplexados, estando apenas um aceso de cada vez.

Cada header define as funções e/ou variáveis que o código com o mesmo nome irá utilizar.

Console.C

```
#include "Console.h"
#include "SystemProfile.h"
#include "Compiler.h"
#include "GenericTypeDefs.h"
#if defined(ENABLE_CONSOLE)
ROM unsigned char CharacterArray[]={'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
#define USART_USE_BRGH_HIGH
#if defined(USART_USE_BRGH_LOW)
    #define SPBRG_VAL    ( ((CLOCK_FREQ/BAUD_RATE)/64) - 1)
#else
    #define SPBRG_VAL    ( ((CLOCK_FREQ/BAUD_RATE)/16) - 1)
#endif

#if SPBRG_VAL > 255
    #error "Calculated SPBRG value is out of range for currnet CLOCK_FREQ."
#endif

//This function will configure the UART for use at in 8 bits, 1 stop, no flowcontrol mode
void ConsoleInit(void)
{
#if defined(USART_USE_BRGH_HIGH)
    TXSTA = 0x24;
#else
    TXSTA = 0x20;
#endif

    RCSTA = 0x90; // 0b10010000;
    SPBRG = SPBRG_VAL;
}

//This function will print the inputed ROM string
void ConsolePutROMString(ROM char* str)
{
    BYTE c;

    while( c = *str++ )
        ConsolePut(c);
}

//This function will print the inputed character
void ConsolePut(BYTE c)
{
    while( !ConsoleIsPutReady() );
    TXREG = c;
}

//This function will print the inputed BYTE to the console in hexidecimal form
void PrintChar(BYTE toPrint)
{
    BYTE PRINT_VAR;
    PRINT_VAR = toPrint;
    toPrint = (toPrint>>4)&0x0F;
    ConsolePut(CharacterArray[toPrint]);
    toPrint = (PRINT_VAR)&0x0F;
    ConsolePut(CharacterArray[toPrint]);
    return;
}

//This function will print the inputed BYTE to the console in decimal form
void PrintDec(BYTE toPrint)
{
    ConsolePut(CharacterArray[toPrint/10]);
    ConsolePut(CharacterArray[toPrint%10]);
}
#endif  //ENABLE_CONSOLE
```

```
BYTE ConsoleGet(void)
{
    // Clear overrun error if it has occured
    // New bytes cannot be received if the error occurs and isn't cleared
    if(RCSTAbits.OERR)
    {
        RCSTAbits.CREN = 0;    // Disable UART receiver
        RCSTAbits.CREN = 1;    // Enable UART receiver
    }

    return RCREG;
}


BYTE ConsoleGetString(char *buffer, BYTE bufferLen)
{
    BYTE v;
    BYTE count;

    count = 0;
    do
    {
        if ( bufferLen-- == 0 )
            break;

        while( !ConsoleIsGetReady() );

        v = RCREG;

        if ( v == '\r' || v == '\n' )
            break;

        count++;
        *buffer++ = v;
        *buffer = '\0';
    } while(1);
    return count;
}
```

HardwareProfile.C (Placa de Interface)

```
#include "Compiler.h"
#include "SymbolTime.h"
#include "P2PDefs.h"
#include "P2P.h"

#define DEBOUNCE_TIME 0x00008FFF

BOOL PUSH_BUTTON_pressed;
TICK PUSH_BUTTON_press_time;

//This function configures the board for the PICDEM-z MRF24J40 usage
void BoardInit(void)
{
	WDTCONbits.SWDTEN = 0; //disable WDT

	// Switches S2 and S3 are on RB5 and RB4 respectively. We want interrupt-on-change
	INTCON = 0x00;

	// There is no external pull-up resistors on S2 and S3. We will use internal pull-ups.
	// The MRF24J40 is using INT0 for interrupts
	// Enable PORTB internal pullups
	INTCON2 = 0x00;
	INTCON3 = 0x00;

	// Make PORTB as input - this is the RESET default
	TRISB = 0xff;

	// Set PORTC control signal direction and initial states
	// disable chip select
	LATC = 0xfd;

	// Set the SPI module for use by Stack
	TRISC = 0xD0;

	// Set the SPI module
	SSPSTAT = 0xC0;
	SSPCON1 = 0x20;

	// D1 and D2 are on RA0 and RA1 respectively, and CS of TC77 is on RA2.
	// Make PORTA as digital I/O.
	// The TC77 temp sensor CS is on RA2.
	ADCON1 = 0x0F;

	// Deselect TC77 (RA2)
	LATA = 0x04;

	// Make RA0, RA1, RA2 and RA4 as outputs.
	TRISA = 0xF8;

	PHY_CS = 1;		//deselect the MRF24J40
	PHY_CS_TRIS = 0;	//make chip select an output

	RFIF = 0;		//clear the interrupt flag

	RCONbits.IPEN = 1;

	INTCON2bits.INTEDG0 = 0;
}

//Byte to indicate which button has been pressed. Return 0 if no button pressed.
BYTE ButtonPressed(void)
{

   TICK tickDifference;

   if(PUSH_BUTTON_1 == 0)
   {
```

```
        //if the button was previously not pressed
        if(PUSH_BUTTON_pressed == FALSE)
        {
            PUSH_BUTTON_pressed = TRUE;
            PUSH_BUTTON_press_time = TickGet();
            return 1;
        }
    }
    else if(PUSH_BUTTON_2 == 0)
    {
        //if the button was previously not pressed
        if(PUSH_BUTTON_pressed == FALSE)
        {
            PUSH_BUTTON_pressed = TRUE;
            PUSH_BUTTON_press_time = TickGet();
            return 2;
        }
    }
    else
    {
        //get the current time
        TICK t = TickGet();

        //if the button has been released long enough
        tickDifference.Val = TickGetDiff(t,PUSH_BUTTON_press_time);

        //then we can mark it as not pressed
        if(tickDifference.Val > DEBOUNCE_TIME)
        {
            PUSH_BUTTON_pressed = FALSE;
        }
    }

    return 0;
}
```

HardwareProfile.C (Contador de Tempo de Bilhar)

```
#include "Compiler.h"
#include "SymbolTime.h"
#include "P2PDefs.h"
#include "P2P.h"

#define DEBOUNCE_TIME 0x00008FFF

BOOL PUSH_BUTTON_pressed;
TICK PUSH_BUTTON_press_time;

//This function configures the board for the PICDEM-z MRF24J40 usage
void BoardInit(void)
{
	WDTCONbits.SWDTEN = 0; //disable WDT

	// activa as interrupções e o int0
	INTCON = 0b11010000;
	INTCON2=0x00;
	INTCON3 = 0x00;



	// Set the SPI module
	SSPSTAT = 0xC0;
	SSPCON1 = 0x20;

			ADCON0=0x00;
			//RA0 -> analogico e o resto digital
	ADCON1 = 0x07;		//0E


	// apenas o 2 e 3 sao entradas
	  TRISA = 0b00001100;
			LATA=0x00;
			PORTA=0x00;

	// Make PORTB as input - this is the RESET default
	TRISB = 0xff;

	// Set PORTC control signal direction and initial states
	// disable chip select
	LATC = 0xfd;

	// Set the SPI module for use by Stack
	TRISC = 0xD0;

			TRISD=0x00;

			TRISE=0x00;

	PHY_CS = 1;		//deselect the MRF24J40
	PHY_CS_TRIS = 0;	//make chip select an output

	RFIF = 0;		//clear the interrupt flag

	RCONbits.IPEN = 1;			//activa a prioridade de interrupções

	INTCON2bits.INTEDG0 = 0;
}

//Byte to indicate which button has been pressed. Return 0 if no button pressed.
BYTE ButtonPressed(void)
{

   TICK tickDifference;

   if(PUSH_BUTTON_1 == 0)
```

```
    {
        //if the button was previously not pressed
        if(PUSH_BUTTON_pressed == FALSE)
        {
            PUSH_BUTTON_pressed = TRUE;
            PUSH_BUTTON_press_time = TickGet();
            return 1;
        }
    }
    else if(PUSH_BUTTON_2 == 0)
    {
        //if the button was previously not pressed
        if(PUSH_BUTTON_pressed == FALSE)
        {
            PUSH_BUTTON_pressed = TRUE;
            PUSH_BUTTON_press_time = TickGet();
            return 2;
        }
    }
    else
    {
        //get the current time
        TICK t = TickGet();

        //if the button has been released long enough
        tickDifference.Val = TickGetDiff(t,PUSH_BUTTON_press_time);

        //then we can mark it as not pressed
        if(tickDifference.Val > DEBOUNCE_TIME)
        {
            PUSH_BUTTON_pressed = FALSE;
        }
    }

    return 0;
}
```

MSPI.C

```
#include "Compiler.h"
#include "GenericTypeDefs.h"

//This function will send a byte over the SPI
 void SPIPut(BYTE v)
{
    PIR1bits.SSPIF = 0;
    do
    {
        SSPCON1bits.WCOL = 0;
        SSPBUF = v;
    } while( SSPCON1bits.WCOL );

    while( PIR1bits.SSPIF == 0 );
}

//This function will read a byte over the SPI
BYTE SPIGet(void)
{
    SPIPut(0x00);
    return SSPBUF;
}
```

P2P.c

```
#include "MSPI.h"
#include "P2P.h"
#include "MRF24J40.h"
#include "Compiler.h"
#include "GenericTypeDefs.h"
#include "Console.h"
#include "SymbolTime.h"
#include "P2PDefs.h"

/************************ VARIABLES ********************************/
#pragma romdata longAddressLocation = 0x0E
    ROM unsigned char myLongAddress[8] = {EUI_0,EUI_1,EUI_2,EUI_3,EUI_4,EUI_5,EUI_6,EUI_7}; // The extended long
address
                                                                    // for the device
#pragma romdata

#ifdef ENABLE_SECURITY
    #pragma romdata securityKey = 0x2E
        ROM unsigned char mySecurityKey[16] = {SECURITY_KEY_00, SECURITY_KEY_01, SECURITY_KEY_02,
SECURITY_KEY_03, SECURITY_KEY_04,
                SECURITY_KEY_05, SECURITY_KEY_06, SECURITY_KEY_07, SECURITY_KEY_08, SECURITY_KEY_09,
SECURITY_KEY_10, SECURITY_KEY_11,
                SECURITY_KEY_12, SECURITY_KEY_13, SECURITY_KEY_14, SECURITY_KEY_15}; // The 16-byte security
key used in the
                                                                    // security module.
    #pragma romdata
    ROM unsigned char mySecurityLevel = SECURITY_LEVEL;             // The security level used in the security module.
                                                // The security level is defined in IEEE 802.15.4 specification
                                                // It can be from 1 to 7.
    ROM unsigned char myKeySequenceNumber = KEY_SEQUENCE_NUMBER;    // The sequence number of security key.
Used to identify
                                                // the security key

            DWORD_VAL OutgoingFrameCounter;
#endif

#if defined(ENABLE_ED_SCAN) || defined(ENABLE_ACTIVE_SCAN) || defined(ENABLE_FREQUENCY_AGILITY)
    // Scan Duration formula for P2P Connection:
    //  60 * (2 ^ n + 1) symbols
    #define SCAN_DURATION_0 SYMBOLS_TO_TICKS(120)
    #define SCAN_DURATION_1 SYMBOLS_TO_TICKS(180)
    #define SCAN_DURATION_2 SYMBOLS_TO_TICKS(300)
    #define SCAN_DURATION_3 SYMBOLS_TO_TICKS(540)
    #define SCAN_DURATION_4 SYMBOLS_TO_TICKS(1020)
    #define SCAN_DURATION_5 SYMBOLS_TO_TICKS(1980)
    #define SCAN_DURATION_6 SYMBOLS_TO_TICKS(3900)
    #define SCAN_DURATION_7 SYMBOLS_TO_TICKS(7740)
    #define SCAN_DURATION_8 SYMBOLS_TO_TICKS(15420)
    #define SCAN_DURATION_9 SYMBOLS_TO_TICKS(30780)
    #define SCAN_DURATION_10 SYMBOLS_TO_TICKS(61500)
    #define SCAN_DURATION_11 SYMBOLS_TO_TICKS(122940)
    #define SCAN_DURATION_12 SYMBOLS_TO_TICKS(245820)
    #define SCAN_DURATION_13 SYMBOLS_TO_TICKS(491580)
    #define SCAN_DURATION_14 SYMBOLS_TO_TICKS(983100)
    const ROM DWORD ScanTime[15] = {SCAN_DURATION_0,SCAN_DURATION_1,SCAN_DURATION_2,SCAN_DURATION_3,

SCAN_DURATION_4,SCAN_DURATION_5,SCAN_DURATION_6,SCAN_DURATION_7,SCAN_DURATION_8,SCAN_DURATION_9,
        SCAN_DURATION_10, SCAN_DURATION_11, SCAN_DURATION_12, SCAN_DURATION_13,SCAN_DURATION_14
    };
#endif
#pragma udata TRX_BUFFER=0x100
BYTE RxBuffer[RX_BUFFER_SIZE];
BYTE TxBuffer[TX_BUFFER_SIZE];
#pragma udata
#pragma udata INDIRECT_BUFFER=0x200
#ifdef ENABLE_INDIRECT_MESSAGE
    INDIRECT_MESSAGE indirectMessages[INDIRECT_MESSAGE_SIZE];   // structure to store the indirect messages
                                                // for nodes with radio off duing idle time
```

```
#endif
#pragma udata
BYTE TxData;
BYTE IEEESeqNum;
#pragma udata BIGvariables1 = 0x300
P2P_CONNECTION_ENTRY      P2PConnections[P2P_CONNECTION_SIZE];          // The peer device records for P2P
connections
#pragma udata
WORD_VAL myPANID;                // the PAN Identifier for the device
volatile BYTE failureCounter = 0;
BYTE currentChannel;             // current operating channel for the device
RECEIVED_FRAME rxFrame;           // structure to store information for the received packet
BYTE    LatestConnection;
volatile P2P_STATUS P2PStatus;
extern BYTE AdditionalConnectionPayload[];  // the additional information regarding the device
                                  // that would like to share with the peer on the
                                  // other side of P2P connection. This information
                                  // is applicaiton specific.
extern BYTE myChannel;
BYTE AddConnection(void);
BOOL isSameAddress(INPUT BYTE *Address1, INPUT BYTE *Address2);
BOOL SendPacket(INPUT BOOL Broadcast, INPUT WORD_VAL DestinationPANID,
            INPUT BYTE *DestinationAddress, INPUT BOOL isCommand,
            INPUT BOOL SecurityEnabled);
#ifdef ENABLE_ACTIVE_SCAN
   BYTE    ActiveScanResultIndex;
   ACTIVE_SCAN_RESULT ActiveScanResults[ACTIVE_SCAN_RESULT_SIZE];  // The results for active scan, including
                                                // the PAN identifier, signal strength and
                                                // operating channel
#endif

#ifdef VERIFY_TRANSMIT
   BYTE CCAFailureTimes = 0;   // Continuous failure times because of CSMA-CA failure
   BYTE AckFailureTimes = 0;   // Continuous failure times because of no acknowledgement received
#endif

#ifdef ENABLE_FREQUENCY_AGILITY
   void StartChannelHopping(INPUT BYTE OptimalChannel);
#endif

#ifdef ENABLE_SLEEP
   TICK DataRequestTimer;
#endif

//This function maintains the operation of the stack. It should be called as often as possible.
void P2PTasks(void)
{
   BYTE i;

   //set the interrupt flag just in case the interrupt was missed
   if(RF_INT_PIN == 0)
   {
      RFIF = 1;
   }

   //If the stack TX has been busy for a long time then
   //time out the TX because we may have missed the interrupt
   //and don't want to lock up the stack forever
   if(P2PStatus.bits.TX_BUSY)
   {
      if(failureCounter == 0x20)
      {
         failureCounter = 0;
         P2PStatus.bits.TX_BUSY = 0;
      }
      else
      {
         failureCounter++;
      }
   }

   #ifdef ENABLE_INDIRECT_MESSAGE
      for(i = 0; i < INDIRECT_MESSAGE_SIZE; i++)
      {
         if( indirectMessages[i].flags.bits.isValid )
```

```
        {
            TICK currentTick = TickGet();
            if( TickGetDiff(currentTick, indirectMessages[i].TickStart) > INDIRECT_MESSAGE_TIMEOUT * ONE_SECOND )
            {
                indirectMessages[i].flags.Val = 0x00;
                printf("\r\nIndirect message expired");
            }
        }
    }
#endif


#ifdef ENABLE_SLEEP
    if( P2PStatus.bits.DataRequesting )
    {
        TICK currentTick = TickGet();
        if( TickGetDiff(currentTick, DataRequestTimer) > RFD_DATA_WAIT )
        {
            printf("\r\nData Request Expired");
            P2PStatus.bits.DataRequesting = 0;
        }
    }
#endif

//if there is a message pending and there isn't one already buffered
if((P2PStatus.bits.RX_PENDING == 1) && (P2PStatus.bits.RX_BUFFERED == 0))
{
    while( P2PStatus.bits.TX_BUSY )
    {
        if(RF_INT_PIN == 0)
        {
            RFIF = 1;
        }
    }

    //disable the RF interrupt
    RFIE = 0;
    // disable the radio to receive new
    // packet
    PHYSetShortRAMAddr(WRITE_BBREG1, 0x04);

    // bypass MRF24J40 errata 5
    //PHYSetShortRAMAddr(WRITE_FFOEN, 0x08);

    //get the size of the packet
    //2 more bytes for RSSI and LQI reading
    RxBuffer[0] = PHYGetLongRAMAddr(0x300) + 2;

    if(RxBuffer[0]<RX_BUFFER_SIZE)
    {
        //disable the ACKs until the buffer is cleared
        PHYSetShortRAMAddr(WRITE_RXMCR,0x20);

        //indicate that data is now stored in the buffer
        P2PStatus.bits.RX_BUFFERED = 1;

        //Disable RX of packets until this one is cleared
        P2PStatus.bits.RX_ENABLED = 0;
        //Clear the pending flag indicating that we removed the packet from the buffer
        P2PStatus.bits.RX_PENDING = 0;

        //copy all of the data from the FIFO into the TxBuffer, plus RSSI and LQI
        for(i=1;i<=RxBuffer[0]+2;i++)
        {
            RxBuffer[i] = PHYGetLongRAMAddr(0x300+i);
        }
    }
    else
    {
        //else it was a larger packet than we can receive
        //flush it
        PHYSetShortRAMAddr(WRITE_RXFLUSH,0x01);
        P2PStatus.bits.RX_PENDING = 0;
    }
```

```
    RFIE = 1;

    // bypass MRF24J40 errata 5
    //PHYSetShortRAMAddr(WRITE_FFOEN, 0x88);

    // enable radio to receive next packet
    PHYSetShortRAMAddr(WRITE_BBREG1, 0x00);
}

if(P2PStatus.bits.RX_BUFFERED == 1)
{
    BYTE addrMode;

    rxFrame.flags.Val = 0;
    //Determine the start of the MAC payload
    addrMode = RxBuffer[2] & 0xCC;
    switch(addrMode)
    {
        case 0xC8: //short dest, long source
            // for P2P only broadcast allows short destination address
            if( RxBuffer[6] == 0xFF && RxBuffer[7] == 0xFF )
            {
                rxFrame.flags.bits.broadcast = 1;
                #ifndef TARGET_SMALL
                    if( RxBuffer[1]&0x40 ) // check if it is intraPAN
                #endif
                {
                    rxFrame.flags.bits.intraPAN = 1;
                    #ifndef TARGET_SMALL
                        rxFrame.SourcePANID.v[0] = RxBuffer[4];
                        rxFrame.SourcePANID.v[1] = RxBuffer[5];
                    #endif
                    for(i = 0; i < 8; i++)
                    {
                        rxFrame.SourceLongAddress[i] = RxBuffer[8 + i];
                    }

                    rxFrame.PayLoadSize = RxBuffer[0] - 19;
                    rxFrame.PayLoad = &(RxBuffer[16]);
                }
                #ifndef TARGET_SMALL
                    else
                    {
                        rxFrame.SourcePANID.v[0] = RxBuffer[8];
                        rxFrame.SourcePANID.v[1] = RxBuffer[9];
                        for(i = 0; i < 8; i++)
                        {
                            rxFrame.SourceLongAddress[i] = RxBuffer[10 + i];
                        }
                        rxFrame.PayLoadSize = RxBuffer[0] - 21;
                        rxFrame.PayLoad = &(RxBuffer[18]);
                    }
                #endif
            }
            break;

        case 0xCC: // long dest, long source
            #ifndef TARGET_SMALL
                if( RxBuffer[1] & 0x40 ) // check if it is intraPAN
            #endif
            {
                rxFrame.flags.bits.intraPAN = 1;
                #ifndef TARGET_SMALL
                    rxFrame.SourcePANID.v[0] = RxBuffer[4];
                    rxFrame.SourcePANID.v[1] = RxBuffer[5];
                #endif
                for(i = 0; i < 8; i++)
                {
                    rxFrame.SourceLongAddress[i] = RxBuffer[14 + i];
                }

                rxFrame.PayLoadSize = RxBuffer[0] - 25;
                rxFrame.PayLoad = &(RxBuffer[22]);
            }
            #ifndef TARGET_SMALL
```

```
            else
            {
                rxFrame.SourcePANID.v[0] = RxBuffer[14];
                rxFrame.SourcePANID.v[1] = RxBuffer[15];
                for(i = 0; i < 8; i++)
                {
                    rxFrame.SourceLongAddress[i] = RxBuffer[16 + i];
                }
                rxFrame.PayLoadSize = RxBuffer[0] - 27;
                rxFrame.PayLoad = &(RxBuffer[24]);
            }
        #endif
        break;
    case 0x88: // short dest, short source
    case 0x80: //dest-none , source-short
    case 0x08: //dest-short, source-none
    case 0x8C: //long dest, short source
        // all other addressing mode will not be supported in P2P
    default:
        // not valid addressing mode or no addressing info
        DiscardPacket();
        return;
}

#ifdef ENABLE_SECURITY
    if( RxBuffer[1] & 0x08 )
    {
        // if security is used, remove the security header and MIC from
        // the payload size
        switch(mySecurityLevel)
        {
            case 0x02:
            case 0x05:
                rxFrame.PayLoadSize -= 21;
                break;
            case 0x03:
            case 0x06:
                rxFrame.PayLoadSize -= 13;
                break;
            case 0x04:
            case 0x07:
                rxFrame.PayLoadSize -= 9;
                break;
            case 0x01:
                rxFrame.PayLoadSize -= 5;
                break;
            default:
                break;
        }
        // remove the security header from the payload
        rxFrame.PayLoad = &(rxFrame.PayLoad[5]);
        rxFrame.flags.bits.security = 1;
    }
#endif
P2PStatus.bits.RxHasUserData = 0;

// check the frame type. Only the data and command frame type
// are supported. Acknowledgement frame type is handled in
// MRF24J40 transceiver hardware.
switch( RxBuffer[1] & 0x07 ) // check frame type
{
    case 0x01:  // data frame
        P2PStatus.bits.RxHasUserData = 1;
        break;
    case 0x03:  // command frame
        rxFrame.flags.bits.commandFrame = 1;
        break;
    default:    // not support frame type
        DiscardPacket();
        return;
}

#ifndef TARGET_SMALL
    rxFrame.PacketLQI = RxBuffer[RxBuffer[0]-1];
    rxFrame.PacketRSSI = RxBuffer[RxBuffer[0]];
```

```
    #endif

    if( P2PStatus.bits.RxHasUserData == 0 )
    {
        // if comes here, we know it is a command frame
        switch( rxFrame.PayLoad[0] )
        {
            case CMD_P2P_CONNECTION_REQUEST:
                {
                    BYTE status = STATUS_SUCCESS;

                    // if a device goes to sleep, it can only have one
                    // connection, as the result, it cannot accept new
                    // connection request
                    #ifdef ENABLE_SLEEP
                        break;
                    #endif

                    // if channel does not math, it may be a
                    // sub-harmonics signal, ignore the request
                    if( currentChannel != rxFrame.PayLoad[1] )
                    {
                        break;
                    }

                    #ifndef TARGET_SMALL
                        // if PANID does not match, ignore the request
                        if( rxFrame.SourcePANID.Val != 0xFFFF &&
                            rxFrame.SourcePANID.Val != myPANID.Val &&
                            rxFrame.PayLoadSize > 2)
                        {
                            status = STATUS_NOT_SAME_PAN;
                        }
                        else
                    #endif
                    {
                        // request accepted, try to add the requesting
                        // device into P2P Connection Entry
                        status = AddConnection();
                    }

                    // if new connection is not allowed, ignore
                    // the request
                    if( P2PStatus.bits.EnableNewConnection == 0 &&
                        status != STATUS_EXISTS &&
                        status != STATUS_ACTIVE_SCAN)
                    {
                        break;
                    }

                    // prepare the P2P_CONNECTION_RESPONSE command
                    FlushTx();
                    WriteData(CMD_P2P_CONNECTION_RESPONSE);
                    WriteData(status);
                    if( status == STATUS_SUCCESS ||
                        status == STATUS_EXISTS )
                    {
                        WriteData(P2P_CAPACITY_INFO);
                        #if ADDITIONAL_CONNECTION_PAYLOAD > 0
                            for(i = 0; i < ADDITIONAL_CONNECTION_PAYLOAD; i++)
                            {
                                WriteData(AdditionalConnectionPayload[i]);
                            }
                        #endif
                    }

                    // unicast the response to the requesting device
                    #ifdef TARGET_SMALL
                        SendPacket(FALSE, myPANID, rxFrame.SourceLongAddress, TRUE, rxFrame.flags.bits.security);
                    #else
                        SendPacket(FALSE,          rxFrame.SourcePANID,          rxFrame.SourceLongAddress,          TRUE,
rxFrame.flags.bits.security);
                    #endif
                }
                break;
```

```
        #ifndef TARGET_SMALL
            case CMD_P2P_CONNECTION_REMOVAL_REQUEST:
                {
                    FlushTx();
                    WriteData(CMD_P2P_CONNECTION_REMOVAL_RESPONSE);

                    for(i = 0; i < P2P_CONNECTION_SIZE; i++)
                    {
                        // if the record is valid
                        if( P2PConnections[i].PeerInfo[0] & 0x80 )
                        {
                            // if the record is the same as the requesting device
                            if( isSameAddress(rxFrame.SourceLongAddress, P2PConnections[i].PeerLongAddress) )
                            {
                                // find the record. disable the record and
                                // set status to be SUCCESS
                                P2PConnections[i].PeerInfo[0] = 0;
                                WriteData(STATUS_SUCCESS);
                                break;
                            }
                        }
                    }

                    if( i == P2P_CONNECTION_SIZE )
                    {
                        // not found, the requesting device is not my peer
                        WriteData(STATUS_ENTRY_NOT_EXIST);
                    }
                    #ifdef TARGET_SMALL
                        SendPacket(FALSE, myPANID, rxFrame.SourceLongAddress, TRUE, rxFrame.flags.bits.security);
                    #else
                        SendPacket(FALSE,  rxFrame.SourcePANID,  rxFrame.SourceLongAddress,  TRUE,
rxFrame.flags.bits.security);
                    #endif
                }
                break;
        #endif

        case CMD_P2P_CONNECTION_RESPONSE:
            {
                switch( rxFrame.PayLoad[1] )
                {
                    case STATUS_SUCCESS:
                    case STATUS_EXISTS:
                        AddConnection();
                        break;

                    case STATUS_ACTIVE_SCAN:
                        if( P2PStatus.bits.Resync )
                        {
                            P2PStatus.bits.Resync = 0;
                        }
                        #ifdef ENABLE_ACTIVE_SCAN
                            else
                            {
                                for(i = 0; i < ActiveScanResultIndex; i++)
                                {
                                    if( ActiveScanResults[ActiveScanResultIndex].Channel == (currentChannel + 11) &&
                                        ActiveScanResults[ActiveScanResultIndex].PANID.Val == rxFrame.SourcePANID.Val )
                                    {
                                        if( ActiveScanResults[ActiveScanResultIndex].RSSIValue < rxFrame.PacketRSSI )
                                        {
                                            ActiveScanResults[ActiveScanResultIndex].RSSIValue = rxFrame.PacketRSSI;
                                        }
                                        break;
                                    }
                                }
                                if( i == ActiveScanResultIndex && (i < ACTIVE_SCAN_RESULT_SIZE))
                                {
                                    ActiveScanResults[ActiveScanResultIndex].Channel = currentChannel;
                                    ActiveScanResults[ActiveScanResultIndex].RSSIValue = rxFrame.PacketRSSI;
                                    ActiveScanResults[ActiveScanResultIndex].PANID.Val = rxFrame.SourcePANID.Val;
                                    ActiveScanResultIndex++;
                                }
```

```
                    }
                #endif
                break;

            default:
                break;
        }

    }
    break;

#ifndef TARGET_SMALL
    case CMD_P2P_CONNECTION_REMOVAL_RESPONSE:
    {
        if( rxFrame.PayLoad[1] == STATUS_SUCCESS )
        {
            for(i = 0; i < P2P_CONNECTION_SIZE; i++)
            {
                // if the record is valid
                if( P2PConnections[i].PeerInfo[0] & 0x80 )
                {
                    // if the record address is the same as the requesting device
                    if( isSameAddress(rxFrame.SourceLongAddress, P2PConnections[i].PeerLongAddress) )
                    {
                        // invalidate the record
                        P2PConnections[i].PeerInfo[0] = 0;
                        break;
                    }
                }
            }
        }
    }

    break;
#endif

#ifdef ENABLE_INDIRECT_MESSAGE
    case CMD_DATA_REQUEST:
    case CMD_MAC_DATA_REQUEST:
        FlushTx();
        for(i = 0; i < INDIRECT_MESSAGE_SIZE; i++)
        {
            if( indirectMessages[i].flags.bits.isValid )
            {
                BYTE j;

                #ifdef ENABLE_BROADCAST
                    if( indirectMessages[i].flags.bits.isBroadcast )
                    {
                        for(j = 0; j < P2P_CONNECTION_SIZE; j++)
                        {
                            if( indirectMessages[i].DestAddress.DestIndex[j] != 0xFF &&

isSameAddress(P2PConnections[indirectMessages[i].DestAddress.DestIndex[j]].PeerLongAddress,
rxFrame.SourceLongAddress) )
                            {
                                indirectMessages[i].DestAddress.DestIndex[j] = 0xFF;
                                for(j = 0; j < indirectMessages[i].PayLoadSize; j++)
                                {
                                    WriteData(indirectMessages[i].PayLoad[j]);
                                }
                                SendPacket(FALSE,     indirectMessages[i].DestPANID,     rxFrame.SourceLongAddress,
indirectMessages[i].flags.bits.isCommand, indirectMessages[i].flags.bits.isSecured);
                                goto DiscardPacketHere;
                            }
                        }
                    }
                    else
                #endif
                if(                                 isSameAddress(indirectMessages[i].DestAddress.DestLongAddress,
rxFrame.SourceLongAddress) )
                {
                    for(j = 0; j < indirectMessages[i].PayLoadSize; j++)
                    {
                        WriteData(indirectMessages[i].PayLoad[j]);
```

```
                        }
                        SendPacket(FALSE,                                   indirectMessages[i].DestPANID,
indirectMessages[i].DestAddress.DestLongAddress,                (BOOL)indirectMessages[i].flags.bits.isCommand,
(BOOL)indirectMessages[i].flags.bits.isSecured);
                        indirectMessages[i].flags.Val = 0;
                        break;
                    }
                }
            }

            if( i == INDIRECT_MESSAGE_SIZE )
            {
                #ifdef TARGET_SMALL
                    SendPacket(FALSE, myPANID, rxFrame.SourceLongAddress, FALSE, FALSE);
                #else
                    SendPacket(FALSE, rxFrame.SourcePANID, rxFrame.SourceLongAddress, FALSE, FALSE);
                #endif
            }
            break;
        #endif

        #if defined(ENABLE_FREQUENCY_AGILITY)
            case CMD_CHANNEL_HOPPING:
                if( rxFrame.PayLoad[1] != currentChannel )
                {
                    break;
                }
                StartChannelHopping(rxFrame.PayLoad[2]);
                printf("\r\nHopping Channel to ");
                PrintDec((currentChannel>>4) + 11);
                break;

        #endif

        default:
            // let upper application layer to handle undefined command frame
            P2PStatus.bits.RxHasUserData = 1;
            break;
    }
  }

  #ifdef ENABLE_SLEEP
    if( P2PStatus.bits.DataRequesting && P2PStatus.bits.RxHasUserData )
    {
      P2PStatus.bits.DataRequesting = 0;
    }
  #endif

  if( rxFrame.PayLoadSize == 0 )
  {
    P2PStatus.bits.RxHasUserData = 0;
  }

DiscardPacketHere:
  if(P2PStatus.bits.RxHasUserData == 0)
  {
    DiscardPacket();
  }
 }

}

//This function needs to be called to discard, clear and reset the Rx module of the MRF24J40.  This function needs to
be called after the user is done processing a received packet
void DiscardPacket(void)
{
 P2PStatus.bits.RxHasUserData = 0;

 //re-enable the ACKS
 RFIE = 0;
 PHYSetShortRAMAddr(WRITE_RXMCR,0x00);
 P2PStatus.bits.RX_BUFFERED = 0;
 P2PStatus.bits.RX_ENABLED = 1;
 RFIE = 1;
}
```

```
//This function writes a value to a LONG RAM address
void PHYSetLongRAMAddr(INPUT WORD address, INPUT BYTE value)
{
    volatile BYTE tmpRFIE = RFIE;

    RFIE = 0;
    PHY_CS = 0;
    SPIPut((((BYTE)(address>>3))&0x7F)|0x80);
    SPIPut((((BYTE)(address<<5))&0xE0)|0x10);
    SPIPut(value);
    PHY_CS = 1;
    RFIE = tmpRFIE;
}

//This function writes a value to a short RAM address
void PHYSetShortRAMAddr(INPUT BYTE address, INPUT BYTE value)
{
    volatile BYTE tmpRFIE = RFIE;

    RFIE = 0;
    PHY_CS = 0;
    SPIPut(address);
    SPIPut(value);
    PHY_CS = 1;
    RFIE = tmpRFIE;
}

//This function reads a value from a short RAM address
BYTE PHYGetShortRAMAddr(INPUT BYTE address)
{
    BYTE toReturn;
    volatile BYTE tmpRFIE = RFIE;

    RFIE = 0;
    PHY_CS = 0;
    SPIPut(address);
    toReturn = SPIGet();
    PHY_CS = 1;
    RFIE = tmpRFIE;

    return toReturn;
}

//This function reads a value from a long RAM address
BYTE PHYGetLongRAMAddr(INPUT WORD address)
{
    BYTE toReturn;
    volatile BYTE tmpRFIE = RFIE;

    RFIE = 0;
    PHY_CS = 0;
    SPIPut(((address>>3)&0x7F)|0x80);
    SPIPut(((address<<5)&0xE0));
    toReturn = SPIGet();
    PHY_CS = 1;
    RFIE = tmpRFIE;

    return toReturn;
}

//This function sets the current operating channel of the MRF24J40
void SetChannel(INPUT BYTE channel)
{
    currentChannel = channel;
    PHYSetLongRAMAddr(RFCTRL0,channel|0x02);
    PHYSetShortRAMAddr(WRITE_RFCTL,0x04);
    PHYSetShortRAMAddr(WRITE_RFCTL,0x00);
}

//This function initializes the MRF24J40 and is required before stack operation is available
void initMRF24J40(void)
{
    BYTE i;
    WORD j;
```

```
PHY_RESETn = 0;
for(j=0;j<(WORD)300;j++){}

PHY_RESETn = 1;
for(j=0;j<(WORD)300;j++){}

/* do a soft reset */
PHYSetShortRAMAddr(WRITE_SOFTRST,0x07);
do
{
   i = PHYGetShortRAMAddr(READ_SOFTRST);
}
while((i&0x07) != (BYTE)0x00);

for(j=0;j<(WORD)1000;j++){}

/* flush the RX fifo */
PHYSetShortRAMAddr(WRITE_RXFLUSH,0x01);

/* Program the short MAC Address, 0xffff */
PHYSetShortRAMAddr(WRITE_SADRL,0xFF);
PHYSetShortRAMAddr(WRITE_SADRH,0xFF);
PHYSetShortRAMAddr(WRITE_PANIDL,myPANID.v[0]);
PHYSetShortRAMAddr(WRITE_PANIDH,myPANID.v[1]);

/* Program Long MAC Address*/
for(i=0;i<(BYTE)8;i++)
{
   PHYSetShortRAMAddr(WRITE_EADR0+i*2,myLongAddress[i]);
}

/* select the correct channel */
PHYSetLongRAMAddr(RFCTRL0,0x02);

/* setup */
PHYSetLongRAMAddr(RFCTRL2,0x80);

// power level to be 0dBms
PHYSetLongRAMAddr(RFCTRL3,0x00);
/* program RSSI ADC with 2.5 MHz clock */
PHYSetLongRAMAddr(RFCTRL6,0x90);

PHYSetLongRAMAddr(RFCTRL7,0x80);

PHYSetLongRAMAddr(RFCTRL8,0x10);

PHYSetLongRAMAddr(SCLKDIV, 0x01);

/* Program CCA mode using RSSI */
PHYSetShortRAMAddr(WRITE_BBREG2,0x80);
/* Enable the packet RSSI */
PHYSetShortRAMAddr(WRITE_BBREG6,0x40);
/* Program CCA, RSSI threshold values */
PHYSetShortRAMAddr(WRITE_RSSITHCCA,0x60);

PHYSetShortRAMAddr(WRITE_FFOEN, 0x98);
PHYSetShortRAMAddr(WRITE_TXPEMISP, 0x95);

do
{
   i = PHYGetLongRAMAddr(RFSTATE);
}
while((i&0xA0) != 0xA0);

PHYSetShortRAMAddr(WRITE_INTMSK,0xE6);

#ifdef ENABLE_INDIRECT_MESSAGE
   PHYSetShortRAMAddr(WRITE_ACKTMOUT, 0xB9);
#endif

// bypass the errata to make RF communication stable
PHYSetLongRAMAddr(RFCTRL1, 0x01);

// Define TURBO_MODE if more bandwidth if required
```

```
    // to enable radio to operate to TX/RX maximum
    // 625Kbps
    #ifdef TURBO_MODE

        PHYSetShortRAMAddr(WRITE_BBREG0, 0x01);
        PHYSetShortRAMAddr(WRITE_BBREG3, 0x38);
        PHYSetShortRAMAddr(WRITE_BBREG4, 0x5C);

        PHYSetShortRAMAddr(WRITE_RFCTL,0x04);
        PHYSetShortRAMAddr(WRITE_RFCTL,0x00);

    #endif
}

//This function initializes the P2P stack and is required before stack operation is available
 void P2PInit(void)
{
    BYTE i;

    myPANID.Val = MY_PAN_ID;

    initMRF24J40();

    //clear all status bits
    P2PStatus.Val = 0;
    //enable reception
    P2PStatus.bits.RX_ENABLED = 1;
    //P2PStatus.bits.RX_BUFFERED = 0;
    P2PStatus.bits.AckRequired = 1;

    for(i = 0; i < P2P_CONNECTION_SIZE; i++)
    {
        P2PConnections[i].PeerInfo[0] = 0;
    }

    InitSymbolTimer();

    TxData = 0;

    IEEESeqNum = TMRL;

    RFIF = 0;
    RFIE = 1;

    if(RF_INT_PIN == 0)
    {
        RFIF = 1;
    }

    #ifdef ENABLE_INDIRECT_MESSAGE
        for(i = 0; i < INDIRECT_MESSAGE_SIZE; i++)
        {
            indirectMessages[i].flags.Val = 0;
        }
    #endif

    #ifdef ENABLE_SECURITY
            OutgoingFrameCounter.Val = 1;
    #endif
}

#ifdef ENABLE_SLEEP
    //Put the MRF24J40 radio into sleep
    void MRF24J40Sleep(void)
    {
        P2PStatus.bits.PHY_SLEEPING = 1;

        //;clear the WAKE pin in order to allow the device to go to sleep
        PHY_WAKE = 0;

        // make a power management reset to ensure device goes to sleep
        PHYSetShortRAMAddr(WRITE_SOFTRST, 0x04);

        //;write the registers required to place the device in sleep
        PHYSetShortRAMAddr(WRITE_TXBCNINTL,0x80);
```

```
    PHYSetShortRAMAddr(WRITE_RXFLUSH,0x60);
    PHYSetShortRAMAddr(WRITE_SLPACK,0x80);
  }


        //This function wakes up the MRF24J40 radio
  void MRF24J40Wake(void)
  {
    BYTE results;
    TICK failureTimer;
    TICK currentDifference;

    //wake up the device
    PHY_WAKE = 1;

    failureTimer = TickGet();

    while(1)
    {
      currentDifference = TickGet();

      currentDifference.Val = TickGetDiff(currentDifference,failureTimer);

      // if timeout, assume the device has waken up
      if(currentDifference.Val > 6250)
      {
        break;
      }

      results = PHYGetShortRAMAddr(READ_ISRSTS);
      if((results & 0x40) != 0x00)
      {
        break;
      }
    }

    while(1)
    {
      currentDifference = TickGet();

      currentDifference.Val = TickGetDiff(currentDifference,failureTimer);

      // if timeout, assume the device has waken up
      if(currentDifference.Val > 6250)
      {
        break;
      }

      results = PHYGetLongRAMAddr(RFSTATE);
      if( (results & 0xE0) == 0xA0 )
      {
        break;
      }

    }

    P2PStatus.bits.PHY_SLEEPING = 0;
  }

        //MRF24J40 is initialized and fully waken up
  BOOL CheckForData(void)
  {
    FlushTx();
    WriteData(CMD_MAC_DATA_REQUEST);
    if( UnicastConnection(0, TRUE, FALSE) )
    {
      P2PStatus.bits.DataRequesting = 1;
      DataRequestTimer = TickGet();
      return TRUE;
    }
    return FALSE;
  }
#endif
```

```
//This is the interrupt handler for the MRF24J40 and P2P stack.
#pragma interruptlow HighISR
void HighISR(void)
{
   if(RFIE && RFIF)
   {
      //clear the interrupt flag as soon as possible such that another interrupt can
      //occur quickly.
TryAgain:
      RFIF = 0;

      //create a new scope for the MRF24J40 interrupts so that we can clear the interrupt
      //flag quickly and then handle the interrupt that we have already received
      {
         MRF24J40_IFREG flags;

         //read the interrupt status register to see what caused the interrupt
         flags.Val=PHYGetShortRAMAddr(READ_ISRSTS);

         if(flags.bits.RF_TXIF)
         {
            //if the TX interrupt was triggered
            //clear the busy flag indicating the transmission was complete
            P2PStatus.bits.TX_BUSY = 0;

            failureCounter = 0;

            #ifndef TARGET_SMALL
               //if we were waiting for an ACK
               if(P2PStatus.bits.TX_PENDING_ACK)
               {
                  BYTE_VAL results;

                  //read out the results of the transmission
                  results.Val = PHYGetShortRAMAddr(READ_TXSR);

                  if(results.bits.b0 == 1)
                  {
                     //the transmission wasn't successful and the number
                     //of retries is located in bits 7-6 of TXSR
                     P2PStatus.bits.TX_FAIL = 1;
                     #ifdef ENABLE_FREQUENCY_AGILITY
                        if( results.bits.b5 )
                        {
                           CCAFailureTimes++;
                        }
                        else
                        {
                           AckFailureTimes++;
                        }
                     #endif
                     //ConsolePutROMString((ROM char *)"\r\nfailed to transmit");
                     //PrintChar(results.Val);
                  }
                  else
                  {
                     //transmission successful
                     //clear that I am pending an ACK, already got it
                     P2PStatus.bits.TX_PENDING_ACK = 0;
                     #ifdef ENABLE_FREQUENCY_AGILITY
                        CCAFailureTimes = 0;
                        AckFailureTimes = 0;
                     #endif
                  }
               }
            #endif

         }

         else if(flags.bits.RF_RXIF)
         {
            //if the RX interrupt was triggered
            if(P2PStatus.bits.RX_ENABLED)
            {
               #ifdef ENABLE_SECURITY
```

```
            if( P2PStatus.bits.RX_SECURITY )
            {
                BYTE DecryptionStatus = PHYGetShortRAMAddr(READ_SECISR);

                P2PStatus.bits.RX_SECURITY = 0;
                if( (DecryptionStatus & 0x02) == 0 )
                {
                    // there is no decryption error
                    P2PStatus.bits.RX_PENDING = 1;
                }
                else
                {
                    PHYSetShortRAMAddr(WRITE_RXFLUSH,0x01);
                }
            }
            else
        #endif
        #if defined(ENABLE_SECURITY) && !defined(TARGET_SMALL)
            if( P2PStatus.bits.RX_IGNORE_SECURITY )
            {
                P2PStatus.bits.RX_IGNORE_SECURITY = 0;
                PHYSetShortRAMAddr(WRITE_RXFLUSH,0x01);
            }
            else
        #endif
            //If the part is enabled for receiving packets right now
            //(not pending an ACK)
            //indicate that we have a packet in the buffer pending to
            //be read into the buffer from the FIFO
            P2PStatus.bits.RX_PENDING = 1;
        }
        else
        {
            //else if the RX is not enabled then we need to flush this packet
            //flush the buffer
            PHYSetShortRAMAddr(WRITE_RXFLUSH,0x01);
        }//end of RX_ENABLED check

    } //end of RXIF check
    #ifdef ENABLE_SECURITY
        else if(flags.bits.SECIF)
        {
            BYTE i;

            #ifndef TARGET_SMALL
                BYTE SourceLongAddress[8];
                DWORD_VAL FrameCounter;
                BYTE FrameControl;
                BYTE j;

                // all the code below is to check the frame counter
                // and compare it with the frame counter for the
                // source device in the memory. We do this to avoid
                // repeat attack.

                // bypass MRF24J40 errata 5
                //PHYSetShortRAMAddr(WRITE_FFOEN, 0x08);

                i = 6;
                FrameControl = PHYGetLongRAMAddr(0x301);
                if( (FrameControl & 0x40) == 0 ) // intra PAN?
                {
                    i += 2;
                }

                FrameControl = PHYGetLongRAMAddr(0x302);
                if( (FrameControl & 0x0C) == 0x0C )
                {
                    i += 8;
                }
                else
                {
                    i += 2;
                }
```

```
                // get the source address
                for(j = 0; j < 8; j++)
                {
                    SourceLongAddress[j] = PHYGetLongRAMAddr(0x300 + i + j);
                }

                for(j = 0; j < 4; j++)
                {
                    FrameCounter.v[j] = PHYGetLongRAMAddr(0x308 + i + j);
                }

                // get key sequence number
                j = PHYGetLongRAMAddr(0x30C + i);

                if( j != myKeySequenceNumber )
                {
                    PHYSetShortRAMAddr(WRITE_SECCR0, 0x80); // ignore the packet
                    P2PStatus.bits.RX_IGNORE_SECURITY = 1;
                }
                else
                {
                    for(i = 0; i < P2P_CONNECTION_SIZE; i++)
                    {
                        if( (P2PConnections[i].PeerInfo[0] & 0x80) &&
                            isSameAddress(P2PConnections[i].PeerLongAddress, SourceLongAddress) )
                        {
                            break;
                        }
                    }
                    if( i < P2P_CONNECTION_SIZE )
                    {
                        if( P2PConnections[i].IncomingFrameCounter.Val > FrameCounter.Val )
                        {
                            PHYSetShortRAMAddr(WRITE_SECCR0, 0x80); // ignore the packet
                            P2PStatus.bits.RX_IGNORE_SECURITY = 1;
                        }
                        else
                        {
                            P2PConnections[i].IncomingFrameCounter.Val = FrameCounter.Val;
                        }
                    }
                }

                // bypass MRF24J40 errata 5
                //PHYSetShortRAMAddr(WRITE_FFOEN, 0x88);

                if( P2PStatus.bits.RX_IGNORE_SECURITY == 0 )
            #endif
            {
                // supply the key
                for(i = 0; i < 16; i++)
                {
                    PHYSetLongRAMAddr(0x2B0 + i, mySecurityKey[i]);
                }
                P2PStatus.bits.RX_SECURITY = 1;

                // set security level and trigger the decryption
                PHYSetShortRAMAddr(WRITE_SECCR0, mySecurityLevel << 3 | 0x40);
            }
        }
    #endif
    else if( flags.Val )
    {
        PHYSetShortRAMAddr(WRITE_RXFLUSH,0x01);
    }

    if( flags.Val )
    {
        goto TryAgain;
    }

  } //end of scope of RF interrupt handler
} //end of if(RFIE && RFIF)

#if defined(__18CXX)
```

```
        //check to see if the symbol timer overflowed
        if(INTCONbits.TMR0IF)
        {
            if(INTCONbits.TMR0IE)
            {
                /* there was a timer overflow */
                INTCONbits.TMR0IF = 0;
                timerExtension1++;
                if(timerExtension1 == 0)
                {
                    timerExtension2++;
                }
            }
        }

        if(INTCONbits.RBIE && INTCONbits.RBIF)
        {
            INTCONbits.RBIF = 0;
        }
    #endif
} //end of interrupt handler


#ifdef ENABLE_INDIRECT_MESSAGE

        //This function store the indirect message for node that turns off radio when idle
    BOOL IndirectPacket(INPUT BOOL Broadcast,
                        INPUT WORD_VAL DestinationPANID,
                        INPUT BYTE *DestinationAddress,
                        INPUT BOOL isCommand,
                        INPUT BOOL SecurityEnabled)
    {
        BYTE i;

        #ifndef ENABLE_BROADCAST
            if( Broadcast )
            {
                return FALSE;
            }
        #endif

        // loop through the available indirect message buffer and locate
        // the empty message slot
        for(i = 0; i < INDIRECT_MESSAGE_SIZE; i++)
        {
            if( indirectMessages[i].flags.bits.isValid == 0 )
            {
                BYTE j;

                // store the message
                indirectMessages[i].flags.bits.isValid          = TRUE;
                indirectMessages[i].flags.bits.isBroadcast      = Broadcast;
                indirectMessages[i].flags.bits.isCommand        = isCommand;
                indirectMessages[i].flags.bits.isSecured        = SecurityEnabled;
                indirectMessages[i].DestPANID.Val               = DestinationPANID.Val;
                if( DestinationAddress != NULL )
                {
                    for(j = 0; j < 8; j++)
                    {
                        indirectMessages[i].DestAddress.DestLongAddress[j] = DestinationAddress[j];
                    }
                }
                #ifdef ENABLE_BROADCAST
                    else
                    {
                        BYTE k = 0;

                        for(j = 0; j < P2P_CONNECTION_SIZE; j++)
                        {
                            if( (P2PConnections[j].PeerInfo[0] & 0x83) == 0x82 )
                            {
                                indirectMessages[i].DestAddress.DestIndex[k++] = j;
                            }
                        }
                        for(; k < P2P_CONNECTION_SIZE; k++)
```

```
                {
                    indirectMessages[i].DestAddress.DestIndex[k] = 0xFF;
                }
            }
        #endif
        indirectMessages[i].PayLoadSize = TxData;
        for(j = 0; j < TxData; j++)
        {
            indirectMessages[i].PayLoad[j] = TxBuffer[j];
        }
        indirectMessages[i].TickStart = TickGet();
        return TRUE;
      }
    }
    return FALSE;
  }
#endif

//This function sends the packet
BOOL SendPacket(INPUT BOOL Broadcast,
        INPUT WORD_VAL DestinationPANID,
        INPUT BYTE *DestinationAddress,
        INPUT BOOL isCommand,
        INPUT BOOL SecurityEnabled)
{
  BYTE headerLength;
  BYTE loc = 0;
  BYTE i;
  #ifndef TARGET_SMALL
    BOOL IntraPAN;
  #endif
  #ifdef VERIFY_TRANSMIT
    WORD w;
  #endif

  // wait for the previous transmission finish
  while( P2PStatus.bits.TX_BUSY )
  {
    if(RF_INT_PIN == 0)
    {
      RFIF = 1;
    }
  }

  // set the frame control in variable i
  if( isCommand )
  {
    i = 0x03;
  }
  else
  {
    i = 0x01;
  }

  // decide the header length for different addressing mode
  #ifndef TARGET_SMALL
    if( DestinationPANID.Val == myPANID.Val ) // this is intraPAN
  #endif
  {
    if( Broadcast )
    {
      headerLength = 15;
    }
    else
    {
      headerLength = 21;
    }
    i |= 0x40;
    #ifndef TARGET_SMALL
      IntraPAN = TRUE;
    #endif
  }

  #ifndef TARGET_SMALL
  else
```

```
{
    if( Broadcast )
    {
        headerLength = 17;
    }
    else
    {
        headerLength = 23;
    }
    IntraPAN = FALSE;
}
#endif

#ifdef ENABLE_SECURITY
    if( SecurityEnabled )
    {
        i |= 0x08;
    }
#endif

if( P2PStatus.bits.AckRequired && Broadcast == FALSE )
{
    i |= 0x20;
}

// set header length
PHYSetLongRAMAddr(loc++, headerLength);
// set packet length
#ifdef ENABLE_SECURITY
    if( SecurityEnabled )
    {
        PHYSetLongRAMAddr(loc++, headerLength + TxData + 5);
    }
    else
#endif
{
    PHYSetLongRAMAddr(loc++, headerLength + TxData);
}

// set frame control LSB
PHYSetLongRAMAddr(loc++, i);

// set frame control MSB
if( Broadcast )
{
    PHYSetLongRAMAddr(loc++, 0xC8);       // long source, short destination
}
else
{
    PHYSetLongRAMAddr(loc++, 0xCC);       // long source, long destination
}

// sequence number
PHYSetLongRAMAddr(loc++, IEEESeqNum++);

// destination PANID
PHYSetLongRAMAddr(loc++, DestinationPANID.v[0]);
PHYSetLongRAMAddr(loc++, DestinationPANID.v[1]);

// destination address
if( Broadcast )
{
    PHYSetLongRAMAddr(loc++, 0xFF);
    PHYSetLongRAMAddr(loc++, 0xFF);
}
else
{
    for(i = 0; i < 8; i++)
    {
        PHYSetLongRAMAddr(loc++, DestinationAddress[i]);
    }
}

#ifndef TARGET_SMALL
    // source PANID if necessary
```

```
    if( IntraPAN == FALSE )
    {
        PHYSetLongRAMAddr(loc++, myPANID.v[0]);
        PHYSetLongRAMAddr(loc++, myPANID.v[1]);
    }
#endif

// source address
for(i = 0; i < 8; i++)
{
    PHYSetLongRAMAddr(loc++, myLongAddress[i]);
}

#ifdef ENABLE_SECURITY
    if( SecurityEnabled )
    {
        WORD keyloc = 0x280;
        // fill the additional security aux header
        for(i = 0; i < 4; i++)
        {
            PHYSetLongRAMAddr(loc++, OutgoingFrameCounter.v[i]);
        }
        OutgoingFrameCounter.Val++;
        PHYSetLongRAMAddr(loc++, myKeySequenceNumber);

        // fill the security key
        for(i = 0; i < 16; i++)
        {
            PHYSetLongRAMAddr(0x280 + i, mySecurityKey[i]);
        }

        // set the cipher mode
        PHYSetShortRAMAddr(WRITE_SECCR0, mySecurityLevel);
    }
#endif


// write the payload
for(i = 0; i < TxData; i++)
{
    PHYSetLongRAMAddr(loc++, TxBuffer[i]);
}
TxData = 0;

P2PStatus.bits.TX_BUSY = 1;

// set the trigger value
if( P2PStatus.bits.AckRequired && Broadcast == FALSE )
{
    i = 0x05;
    #ifndef TARGET_SMALL
        P2PStatus.bits.TX_PENDING_ACK = 1;
    #endif
}
else
{
    i = 0x01;
    #ifndef TARGET_SMALL
        P2PStatus.bits.TX_PENDING_ACK = 0;
    #endif
}
#ifdef ENABLE_SECURITY
    if( SecurityEnabled )
    {
        i |= 0x02;
    }
#endif

// now trigger the transmission
PHYSetShortRAMAddr(WRITE_TXNMTRIG, i);

#ifdef VERIFY_TRANSMIT
    w = 0;
    while(1)
    {
```

```
            if( RF_INT_PIN == 0 )
            {
                RFIF = 1;
            }
            if( P2PStatus.bits.TX_BUSY == 0 )
            {
                if( P2PStatus.bits.TX_FAIL )
                {
                    P2PStatus.bits.TX_FAIL = 0;
                    return FALSE;
                }
                break;
            }
            if( ++w > 0xFFFE )
            {
                return FALSE;
            }
        }
        CCAFailureTimes = 0;
        AckFailureTimes = 0;
    #endif

    return TRUE;
}

//This function broadcast a packet
BOOL BroadcastPacket(   INPUT WORD_VAL DestinationPANID,
                        INPUT BOOL isCommand,
                        INPUT BOOL SecurityEnabled )
{
    #ifdef ENABLE_INDIRECT_MESSAGE
        BYTE i;
        for(i = 0; i < P2P_CONNECTION_SIZE; i++)
        {
            if( P2PConnections[i].PeerInfo[0]>= 0x80 ) PrintChar(P2PConnections[i].PeerInfo[0]);

            if( (P2PConnections[i].PeerInfo[0] & 0x83) == 0x82 )
            {
                IndirectPacket(TRUE, DestinationPANID, NULL, isCommand, SecurityEnabled);
            }
        }
    #endif
    return SendPacket(TRUE, DestinationPANID, NULL, isCommand, SecurityEnabled);
}

//This is one of ways to unicast a packet to the peer device
BOOL UnicastConnection( INPUT BYTE ConnectionIndex,
                        INPUT BOOL isCommand,
                        INPUT BOOL SecurityEnabled)
{
    if( P2PConnections[ConnectionIndex].PeerInfo[0] & 0x80 )
    {
        #ifdef ENABLE_INDIRECT_MESSAGE
            // check if RX on when idle
            if( (P2PConnections[ConnectionIndex].PeerInfo[0] & 0x03) == 0x02 )
            {
                return IndirectPacket(FALSE, myPANID, P2PConnections[ConnectionIndex].PeerLongAddress, isCommand,
SecurityEnabled);
            }
        #endif
        return UnicastLongAddress(myPANID, P2PConnections[ConnectionIndex].PeerLongAddress, isCommand,
SecurityEnabled);
    }
    return FALSE;
}

//Unicast a packet to the peer device represented by the destination long address
BOOL UnicastLongAddress(INPUT WORD_VAL DestinationPANID,
                        INPUT BYTE *DestinationAddress,
                        INPUT BOOL isCommand,
                        INPUT BOOL SecurityEnabled)
{
    #ifdef ENABLE_INDIRECT_MESSAGE
        BYTE i;
```

```
        for(i = 0; i < P2P_CONNECTION_SIZE; i++)
        {
            // check if RX on when idle
            if( (P2PConnections[i].PeerInfo[0] & 0x83) == 0x82 &&
                isSameAddress(DestinationAddress, P2PConnections[i].PeerLongAddress) )
            {
                return IndirectPacket(FALSE, DestinationPANID, DestinationAddress, isCommand, SecurityEnabled);
            }
        }
    #endif
    return SendPacket(FALSE, DestinationPANID, DestinationAddress, isCommand, SecurityEnabled);
}

//This function compares two long addresses and returns the boolean to indicate if they are the same
BOOL    isSameAddress(INPUT BYTE *Address1, INPUT BYTE *Address2)
{
    BYTE i;

    for(i = 0; i < 8; i++)
    {
        if( Address1[i] != Address2[i] )
        {
            return FALSE;
        }
    }
    return TRUE;
}

#ifdef ENABLE_ED_SCAN
            //This function create a new P2P connection between two devices
    BYTE CreateNewConnection(INPUT BYTE RetryInterval, INPUT DWORD ChannelMap)
#else
    BYTE CreateNewConnection(INPUT BYTE RetryInterval)
#endif
{
    TICK    startTick = TickGet();
    BYTE    EnableConnection = P2PStatus.bits.EnableNewConnection;
    BYTE    Interval = 0;
    #ifdef ENABLE_ED_SCAN
        DWORD ChannelMask = 0x00000800;
        BYTE i = 0;
    #endif

    EnableNewConnection();
    P2PStatus.bits.SearchConnection = 1;
    while( P2PStatus.bits.SearchConnection )
    {
        TICK currentTick = TickGet();

        if( TickGetDiff(currentTick, startTick) > (ONE_SECOND) )
        {
            startTick = currentTick;

            // avoid multiply of the DWORD, save about 150 byes
            // of programming space for PIC18
            if( ++Interval < RetryInterval )
            {
                continue;
            }
            Interval = 0;

            #ifdef ENABLE_ED_SCAN
                // if enable energy scan, we need to scan all channels
                // here loop through all available channels
                while( (ChannelMap & (ChannelMask << i)) == 0 )
                {
                    i++;
                    if( i > 15 )
                    {
                        i = 0;
                    }
                }
                SetChannel(i << 4);
                printf("\r\nScan channel ");
                PrintDec(i+11);
```

```
            i++;
        #endif

        FlushTx();
        WriteData(CMD_P2P_CONNECTION_REQUEST);
        WriteData(currentChannel);
        WriteData(P2P_CAPACITY_INFO);
        #if ADDITIONAL_CONNECTION_PAYLOAD > 0
            for(i = 0; i < ADDITIONAL_CONNECTION_PAYLOAD; i++)
            {
                WriteData(AdditionalConnectionPayload[i]);
            }
        #endif

        SendPacket(TRUE, myPANID, NULL, TRUE, FALSE);
    }

    P2PTasks();
  }

  P2PStatus.bits.EnableNewConnection = EnableConnection;

  return LatestConnection;
}

//This function returns the boolean to indicate if a new packet has been received by the stack
BOOL ReceivedPacket(void)
{
  P2PTasks();

  return P2PStatus.bits.RxHasUserData;
}

#ifdef ENABLE_DUMP
        //This function prints out the content of the connection  with the input index of the P2P Connection Entry
  void DumpConnection(INPUT BYTE index)
  {
    BYTE i, j;

    if( index < P2P_CONNECTION_SIZE )
    {
        printf("\r\n\r\nConnection     MAC ADRESS     PeerInfo\r\n");
        if( P2PConnections[index].PeerInfo[0] & 0x80 )
        {
            PrintChar(index);
            printf("             ");
            for(i = 0; i < 8; i++)
            {
                PrintChar( P2PConnections[index].PeerLongAddress[7-i] );
            }
            printf("    ");
            #if ADDITIONAL_CONNECTION_PAYLOAD > 0
                for(i = 0; i < ADDITIONAL_CONNECTION_PAYLOAD + 1; i++)
                {
                    PrintChar( P2PConnections[index].PeerInfo[i] );
                }
            #endif
            printf("\r\n");
        }
    }
    else
    {
        printf("\r\n\r\nConnection     PeerLongAddress     PeerInfo\r\n");
        for(i = 0; i < P2P_CONNECTION_SIZE; i++)
        {

            if( P2PConnections[i].PeerInfo[0] & 0x80 )
            {
                PrintChar(i);
                printf("             ");
                for(j = 0; j < 8; j++)
                {
                    PrintChar( P2PConnections[i].PeerLongAddress[7-j] );
                }
                printf("    ");
```

```
            #if ADDITIONAL_CONNECTION_PAYLOAD > 0
               for(j = 0; j < ADDITIONAL_CONNECTION_PAYLOAD + 1; j++)
               {
                  PrintChar( P2PConnections[i].PeerInfo[j] );
               }
            #endif
            printf("\r\n");
         }
      }
   }
}
#endif

//A P2P Connection Request or Response has been received and stored in rxFrame structure
BYTE AddConnection(void)
{
   BYTE i;
   BYTE status = STATUS_SUCCESS;
   BYTE connectionSlot = 0xFF;

   // if no peerinfo attached, this is only an active scan request,
   // so do not save the source device's info
   #ifdef ENABLE_ACTIVE_SCAN
      if( rxFrame.PayLoadSize < 3 )
      {
         return STATUS_ACTIVE_SCAN;
      }
   #endif

   // loop through all entry and locate an proper slot
   for(i = 0; i < P2P_CONNECTION_SIZE; i++)
   {
      // check if the entry is valid
      if( P2PConnections[i].PeerInfo[0] & 0x80 )
      {
         // check if the entry address matches source address of current received packet
         if( isSameAddress(rxFrame.SourceLongAddress, P2PConnections[i].PeerLongAddress) )
         {
            connectionSlot = i;
            status = STATUS_EXISTS;
            break;
         }
      }
      else if( connectionSlot == 0xFF )
      {
         // store the first empty slot
         connectionSlot = i;
      }
   }

   if( connectionSlot == 0xFF )
   {
      return STATUS_NOT_ENOUGH_SPACE;
   }
   else
   {
      if( P2PStatus.bits.EnableNewConnection == 0 )
      {
         return status;
      }

      // store the source address
      for(i = 0; i < 8; i++)
      {
         P2PConnections[connectionSlot].PeerLongAddress[i] = rxFrame.SourceLongAddress[i];
      }

      // store the capacity info and validate the entry
      P2PConnections[connectionSlot].PeerInfo[0] = rxFrame.PayLoad[2] | 0x80;
      // store possible additional connection payload
      #if ADDITIONAL_CONNECTION_PAYLOAD > 0
         for(i = 0; i < ADDITIONAL_CONNECTION_PAYLOAD; i++)
         {
            P2PConnections[connectionSlot].PeerInfo[1+i] = rxFrame.PayLoad[3+i];
         }
```

```
        #endif

        #ifdef ENABLE_SECURITY
            // if security is enabled, clear the incoming frame control
            P2PConnections[connectionSlot].IncomingFrameCounter.Val = 0;
        #endif
        LatestConnection = connectionSlot;
        P2PStatus.bits.SearchConnection = 0;
    }
    return status;
}


#ifdef ENABLE_ACTIVE_SCAN
            //This function do an active scan and returns the number of PANs catched during the scan. The scan results
are stored in global variable ActiveScanResults
    BYTE ActiveScan(INPUT BYTE ScanDuration, INPUT DWORD ChannelMap)
    {
        BYTE i;
        DWORD channelMask = 0x00000800;
        BYTE backupChannel = currentChannel;

        for(i = 0; i < ACTIVE_SCAN_RESULT_SIZE; i++)
        {
            ActiveScanResults[i].Channel = 0xFF;
        }

        ActiveScanResultIndex = 0;
        i = 0;
        while( i < 16 )
        {
            if( ChannelMap & (channelMask << i) )
            {
                TICK ScanStart;
                TICK tempTick;
                WORD_VAL tmpPANID;

                /* choose appropriate channel */
                SetChannel( (i << 4) );

                FlushTx();
                WriteData(CMD_P2P_CONNECTION_REQUEST);
                WriteData(currentChannel);
                tmpPANID.Val = 0xFFFF;
                SendPacket(TRUE, tmpPANID, NULL, TRUE, FALSE);

                ScanStart = TickGet();
                while(1)
                {
                    P2PTasks();

                    tempTick = TickGet();
                    if( TickGetDiff(tempTick, ScanStart) > ((DWORD)(ScanTime[ScanDuration])) )
                    {
                        // if scan time exceed scan duration, prepare to scan the next channel
                        break;
                    }
                }
            }
            i++;
        }

        SetChannel(backupChannel);

        return ActiveScanResultIndex;
    }

#endif

#ifdef ENABLE_ED_SCAN
            //This function finds out the optimal channel with least noise after doing energy scan on all available
channels supplied.
    BYTE OptimalChannel(INPUT BYTE ScanDuration, INPUT DWORD ChannelMap, OUTPUT BYTE *RSSIValue)
    {
        BYTE i;
```

```
BYTE OptimalChannel;
BYTE minRSSI = 0xFF;
DWORD channelMask = 0x00000800;

ConsolePutROMString((ROM char*)"\r\nEnergy Scan Results:");
i = 0;
while( i < 16 )
{
   if( ChannelMap & (channelMask << i) )
   {
      BYTE RSSIcheck;
      BYTE maxRSSI = 0;
      TICK ScanStart;
      TICK tempTick;
      BYTE j, k;

      /* choose appropriate channel */
      SetChannel( (i << 4) );

      ScanStart = TickGet();

      while(1)
      {
         // calculate RSSI for firmware request
         PHYSetShortRAMAddr(WRITE_BBREG6, 0x80);

         // Firmware Request the RSSI
         RSSIcheck = PHYGetShortRAMAddr (READ_BBREG6);
         while ((RSSIcheck & 0x01) != 0x01)
         {
            RSSIcheck = PHYGetShortRAMAddr (READ_BBREG6);
         }

         // read the RSSI
         RSSIcheck = PHYGetLongRAMAddr(0x210);
         if( RSSIcheck > maxRSSI )
         {
            maxRSSI = RSSIcheck;
         }

         tempTick = TickGet();
         if( TickGetDiff(tempTick, ScanStart) > ((DWORD)(ScanTime[ScanDuration])) )
         {
            // if scan time exceed scan duration, prepare to scan the next channel
            break;
         }
      }

      printf("\r\nChannel ");
      PrintDec(i+11);
      printf(": ");
      j = maxRSSI/5;
      for(k = 0; k < j; k++)
      {
         ConsolePut('-');
      }
      printf(" ");
      PrintChar(maxRSSI);

      if( maxRSSI < minRSSI )
      {
         minRSSI = maxRSSI;
         OptimalChannel = (i << 4);
         if( RSSIValue )
         {
            *RSSIValue = minRSSI;
         }
      }
   }
   i++;
}

// enable RSSI attached to received packet again after
// the energy scan is finished
PHYSetShortRAMAddr(WRITE_BBREG6, 0x40);
```

```
    return OptimalChannel;
  }

#endif


#ifdef ENABLE_FREQUENCY_AGILITY
        //This function broadcast the channel hopping command and after that, change operating channel to the input optimal channel
  void StartChannelHopping(INPUT BYTE OptimalChannel)
  {
    BYTE i;

    for( i = 0; i < FA_BROADCAST_TIME; i++)
    {
      TICK startTick = TickGet();
      while(1)
      {
        TICK currentTick = TickGet();
        if( TickGetDiff(currentTick, startTick) > SCAN_DURATION_8 )
        {
          FlushTx();
          WriteData(CMD_CHANNEL_HOPPING);
          WriteData(currentChannel);
          WriteData(OptimalChannel);
          SendPacket(TRUE, myPANID, NULL, TRUE, FALSE);
          break;
        }
      }
    }
    SetChannel(OptimalChannel);
  }

        //This function try to resynchronize the connection with its peer with the input destination address
  BOOL ResyncConnection(INPUT BYTE *DestinationAddress, INPUT DWORD ChannelMap)
  {
    BYTE i;
    BYTE j;
    BYTE backupChannel = currentChannel;
    TICK startTick = TickGet();

    P2PStatus.bits.Resync = 1;
    for(i = 0; i < RESYNC_TIMES; i++)
    {
      DWORD ChannelMask = 0x00000800;

      j = 0;
      while(P2PStatus.bits.Resync)
      {
        TICK currentTick = TickGet();

        if( TickGetDiff(currentTick, startTick) > SCAN_DURATION_8 )
        {
          startTick = currentTick;

          while( (ChannelMap & (ChannelMask << j)) == 0 )
          {
            if( ++j > 15 )
            {
              goto GetOutOfLoop;
            }
          }
          SetChannel(j << 4);
          j++;


          FlushTx();
          WriteData(CMD_P2P_CONNECTION_REQUEST);
          WriteData(currentChannel);

          SendPacket(FALSE, myPANID, DestinationAddress, TRUE, FALSE);
        }
        P2PTasks();
      }
```

```
            printf("\r\nResynchronized Connection to Channel ");
            PrintDec((currentChannel>>4)+11);
            return TRUE;
GetOutOfLoop:
            Nop();
        }
        SetChannel(backupChannel);
        P2PStatus.bits.Resync = 0;
        return FALSE;
    }

    #ifdef FREQUENCY_AGILITY_STARTER
                        //This function try to do energy scan to locate the channel with least noise. If the channel is not
current operating channel, process of channel hopping will be started.
        BOOL InitChannelHopping( INPUT DWORD ChannelMap)
        {
            BYTE RSSIValue;
            BYTE backupChannel = currentChannel;
            BYTE optimalChannel = OptimalChannel(10, ChannelMap, &RSSIValue);

            SetChannel(backupChannel);
            if( optimalChannel == backupChannel )
            {
                return FALSE;
            }

            printf("\r\nHopping to Channel ");
            PrintDec((optimalChannel>>4)+11);
            StartChannelHopping(optimalChannel);
            return TRUE;
        }
    #endif

#endif



#pragma code highVector=0x08
void HighVector (void)
{
    _asm goto HighISR _endasm
}
#pragma code /* return to default code section */

#pragma code lowhVector=0x18
void LowVector (void)
{
    _asm goto HighISR _endasm
}
#pragma code /* return to default code section */
```

SimpleExampleNode1.c (Placa de Interface)

```
#include "Console.h"
#include "P2P.h"
#include "MRF24J40.h"
#include "SymbolTime.h"
#include <string.h>

#if ADDITIONAL_CONNECTION_PAYLOAD > 0
   BYTE AdditionalConnectionPayload[ADDITIONAL_CONNECTION_PAYLOAD];
#endif

BYTE myChannel = CHANNEL_25;

BYTE c;
unsigned int i,j,tarifa,fraccao,power_down,pass,var_aux,endereco;
unsigned long total_facturado;
char var_recebida[100];
char msgx[]="R";
char msg_vazia[]="";
char msg_test[];
char quer_ler[]="$%";

void main(void)
{
   BYTE i;
   BoardInit();
   ConsoleInit();
   P2PInit();
   INTCONbits.GIEH = 1;
   ConsolePutROMString((ROM char*)"\rIniciar...");
   LED_1 = 0;
   // Set the device to operate on the optimal channel
   SetChannel(myChannel);
   EnableNewConnection();
   i = CreateNewConnection(2);
   #ifdef ENABLE_DUMP
      DumpConnection(i);
   #endif
   // Turn on LED 1 to indicate P2P connection established
   LED_1 = 1;
   while(1)
   {
      if( ReceivedPacket() )
      {
                              ConsolePutROMString((ROM char*)"\n");
         for(i = 0; i < rxFrame.PayLoadSize; i++)
         {
            ConsolePut(rxFrame.PayLoad[i]);
         }
         ConsolePutROMString((ROM char*)"%\r\n");
         // Toggle LED2 to indicate receiving a packet.
         LED_2 ^= 1;

         DiscardPacket();
      }

                  if ( ConsoleIsGetReady())
                  {
                        BYTE x=46;
                        ConsoleGetString(var_recebida,x);
                        FlushTx();
                        for(i = 0; i < 46; i++)
            {
                              WriteData(var_recebida[i]);
                        }
                        BroadcastPacket(myPANID, FALSE, FALSE);
                  }
   }
}
```

SimpleExampleNode2.c (Contador de Tempo de Bilhar)

```
#include "Console.h"
#include "P2P.h"
#include "MRF24J40.h"
#include "SymbolTime.h"
#include "P2PDefs.h"
#include <p18f452.h>
#include <string.h>

#define _GLOBALS_C
#include "SimpleExampleNode2.h"
#include "delays.h"

//funções
void main (void);
void LInterruptHandle (void);
void Grava_EEP(unsigned int eep_endereco, unsigned int eep_valor);  //grava na eeprom
unsigned int LER_EEP(unsigned int eep_endereco);                                    //lê a eeprom
unsigned int Calculo_Tarifa(unsigned int num_tarifa);                              //lê os valores da eeprom e
calcula a tarifa
unsigned int Calculo_Fraccao();
                        //lê o valor da fracção da eeprom
unsigned int mostra_tarifa();
            //função que mostra a tarifa nos displays superiores
unsigned int valor_a_pagar();
            //função que mostra o valor a pagar nos displays inferiores
unsigned char verifica_sensores();
            //função que vê o estado dos sensores: 1111 para fechado e 0000 para aberto
unsigned int programa_tarifa();

//variáveis
unsigned int disp_a_mostrar;            //display a mostrar
unsigned int fraccao;                                   //fracção de contagem
unsigned int n_ciclos_contagem_i;       //numero de ciclos que o timer irá executar até incrementar o valor a pagar
unsigned int ciclo_actual,ciclo_2,ciclo_3,ciclo_4;                  //numero do ciclo em que o timer se encontra
unsigned float erro_contagem;                       //erro de contagem
unsigned float n_ciclos_contagem_f;     //numero de ciclos que o timer teria que executar, mas não pode pois é um
valor real
const float t_interrupcao = 16.384;     //interrupção do timer2 de 16.384ms
char* gaveta;                           //estado da gaveta
char* modo;                                     //modo: normal
unsigned int estado_displays;           //estado dos displays quando está em alarme
unsigned int estado_displays2;                  //estado dos displays quando está em programação
unsigned char estado_sensores;                  //estado dos sensores
unsigned int power_down,power_down_h,power_down_l;
unsigned int password;                                  //password para programação
unsigned int total_facturado_h,total_facturado_m,total_facturado_l,total_facturado_ajuda;                   //total
facturado a gravar na eeprom
unsigned int tarifa_save_h,tarifa_save_l;
unsigned long total_facturado;
unsigned int serial_number_a,serial_number_s,serial_number_n;       //serial number ano/semana/numero
unsigned int u;                         //variavel de contagem (i)
//variaveis para cada display
unsigned int tarifa_d_5,tarifa_d_4,tarifa_d_3,tarifa_d_2,tarifa_d_1;
unsigned int valor_a_pagar_d_5,valor_a_pagar_d_4,valor_a_pagar_d_3,valor_a_pagar_d_2,valor_a_pagar_d_1;

char msgx[]="X";                //separador
char info[20];                          //temporário
char msgsn[]="SN";              //serial number
char msgf[]="F";                //fracção
char msgt01[]="T01";            //tarifa 01
char msgpd[]="PD";              //power down
char msgpw[]="PW";              //password
char msgtf[]="TF";              //total facturado
```

```
char msgci[]="$#";                //codigo inicial
char msgci2[]="$R";               //codigo inicial do LIVE
char msgcf[]="#$";                //codigo final
char msgfg[]="FG";                //fecho de gaveta
char msg_enviar[60];              //mensagem final
char msg_enviar2[30];             //mensagem final
char msg_vazia[]=" ";
char msg_test[];

#if ADDITIONAL_CONNECTION_PAYLOAD > 0
   BYTE AdditionalConnectionPayload[ADDITIONAL_CONNECTION_PAYLOAD];
#endif

BYTE myChannel = CHANNEL_25;

//Low priority interrupt vector
#pragma code LOW_INTERRUPT_VECTOR = 0x000018          // Low priority interrupt vector is there
void low_ISR (void){
  _asm goto LInterruptHandle _endasm
}
#pragma code //Allows the linker to locate the remaining code
void main ()
{
BYTE i, j;
    BoardInit();
   ConsoleInit();
   P2PInit();

            //mete os displays apagados
            disp_a_mostrar=0;
            activa_display(disp_a_mostrar);

            //configuração timer2 para o contador de frações
            RCONbits.IPEN=1;     // Enable priority levels on interrupts
            PIR1=0;              //On met à 0 tous les flags d'interruption
            IPR1=0b00000000;    //define a prioridade baixa do timer2
            PIE1=0b00000010;    //Interruption TMR2 activée
            TMR2=0;
            PIR1bits.TMR2IF=0; //Clear flag
            INTCONbits.GIEH = 1; //Activation des interruptions hautes
            INTCONbits.PEIE = 1; //Peripheral Interrupt Enable bit
            T2CON=0b01111111;                        //cada interrupção do timer é de 16.384ms

            //timer 3 para outras coisas.
            TMR3L=0;
            TMR3H=0;
            IPR2=0b00000000;    //define a prioridade baixa do timer3
            PIE2=0b00000010;
            T3CON=0b00110001;             //cada interrupção é de 131,072ms
   ConsolePutROMString((ROM char*)"\r\nStarting P2P Demo...");
   LED_1_W=0;
            LED_2_W=1;

   // Set default channel
   SetChannel(myChannel);
   EnableNewConnection();
   i = CreateNewConnection(2);
            #ifdef ENABLE_DUMP
      DumpConnection(0xFF);
   #endif
            //vai ler o serial number
            serial_number_a=LER_EEP(EEP_SN_A);
            serial_number_s=LER_EEP(EEP_SN_S);
            serial_number_n=LER_EEP(EEP_SN_N);

            //vai buscar o valor da tarifa às posições de memória
            tarifa=Calculo_Tarifa(1);

            //vai ler o total facturado
            total_facturado=(65536*LER_EEP(EEP_TOTALH))+(256*LER_EEP(EEP_TOTALM))+LER_EEP(EEP_TOTALL);

            //o erro de contagem inicial é zero
            erro_contagem=0;

            //vai ler a fracção à posição de memória
```

```
        fraccao=Calculo_Fraccao();

        //calcula o numero de ciclos da interrupção sem arredondamentos
        n_ciclos_contagem_f=((fraccao*36000)/(tarifa/100))/t_interrupcao;    //3662.109 para 3€

        //arredonda as unidades o numero de ciclos da interrupção
        n_ciclos_contagem_i=n_ciclos_contagem_f;

        //calcula o erro de contagem
        erro_contagem=n_ciclos_contagem_f*t_interrupcao-n_ciclos_contagem_i*t_interrupcao;

        //lê o numero de powerdown até agora:
        power_down_h= LER_EEP(EEP_PWRDWNH);
        power_down_l= LER_EEP(EEP_PWRDWNL);
        power_down=1+power_down_l+(256*power_down_h);
        power_down_h=power_down/256;
        power_down_l=power_down-power_down_h*256;
        Grava_EEP(EEP_PWRDWNL,power_down_l);
        Grava_EEP(EEP_PWRDWNH,power_down_h);


        //calcula a password
        password= LER_EEP(EEP_PASSWH)*65536+LER_EEP(EEP_PASSWM)*256+LER_EEP(EEP_PASSWL);

        //mete o numero de ciclo actual a zero
        ciclo_actual=0;

        ciclo_2=0;
        ciclo_3=0;
        ciclo_4=0;

        //valor a pagar inicial=0
        val_pagar=fraccao;

        //define modo
        gaveta="fechada";
        modo="normal";

        //prepara a string
        strcpy(msg_enviar,msgci);               //mete o código inicial
        strcat(msg_enviar,msgsn);               //mete o SN -> serial number
        itoa((serial_number_a),info);
        strcat(msg_enviar,info);
        strcat(msg_enviar,msgx);                //separa com X
        itoa((serial_number_s),info);
        strcat(msg_enviar,info);
        strcat(msg_enviar,msgx);                //separa com X
        itoa((serial_number_n),info);
        strcat(msg_enviar,info);

        //estado dos displays em alarme
        estado_displays=0;
while (1)

 {
                LED_1_W=1;

                //modo normal é o de contagem
                if ((modo=="normal") || (modo=="rede"))
                {
                        verifica_sensores();
                }

   if( ReceivedPacket() )
    {
                        //se acaba de receber o codigo de leitura dos valores da eeprom
                        if ((rxFrame.PayLoad[0]=='$') && (rxFrame.PayLoad[1]=='%'))
                        {
                                //prepara a string
                                strcpy(msg_enviar,msgci);               //mete o código inicial
                                strcat(msg_enviar,msgsn);               //mete o SN -> serial number
                                        itoa((serial_number_a),info);
                                        strcat(msg_enviar,info);
                                strcat(msg_enviar,msgx);                //separa com X
                                        itoa((serial_number_s),info);
```

```
                strcat(msg_enviar,info);
        strcat(msg_enviar,msgx);                //separa com X
                itoa((serial_number_n),info);
                strcat(msg_enviar,info);
        strcat(msg_enviar,msgx);                //separa com X
        strcat(msg_enviar,msgf);                //mete o F -> fracção
                itoa((fraccao),info);
                strcat(msg_enviar,info);
        strcat(msg_enviar,msgx);                //separa com X
        strcat(msg_enviar,msgpd);               //mete o PD -> power down
                itoa((power_down),info);
                strcat(msg_enviar,info);
        strcat(msg_enviar,msgx);                //separa com X
        strcat(msg_enviar,msgt01);              //mete o T01 -> tarifa 01
                itoa((tarifa),info);
                strcat(msg_enviar,info);
        strcat(msg_enviar,msgx);                //separa com X
        strcat(msg_enviar,msgtf);               //mete o TF -> password
                itoa((total_facturado),info);
                strcat(msg_enviar,info);
        strcat(msg_enviar,msgx);                //separa com X
        strcat(msg_enviar,msgpw);               //mete o PW -> password
                itoa((password),info);
                strcat(msg_enviar,info);
        strcat(msg_enviar,msgcf);               //mete o codigo final

        //envia a string
        FlushTx();
        for(i = 0; i < (strlen(msg_enviar)); i++)
        {
                WriteData(msg_enviar[i]);
        }
        UnicastConnection(0, FALSE, TRUE);
        //BroadcastPacket(myPANID, FALSE, FALSE);

}       //acaba o codigo que faz quando recebe trama de leitura

//*****************************************************
//*****************************************************

//se acaba de receber o codigo de modo rede
if ((rxFrame.PayLoad[0]=='!') && (rxFrame.PayLoad[1]=='R'))
{
        //verifica se é o mesmo numero de ano
        msg_test[0]=rxFrame.PayLoad[2];
        msg_test[1]=rxFrame.PayLoad[3];
        msg_test[2]=rxFrame.PayLoad[4];
        if (atoi(msg_test)==serial_number_a)
        {
                msg_test[0]=rxFrame.PayLoad[5];
                msg_test[1]=rxFrame.PayLoad[6];
                msg_test[2]=rxFrame.PayLoad[7];
                //verifica se é o mesmo numero de semana
                if (atoi(msg_test)==serial_number_s)
                {
                                msg_test[0]=rxFrame.PayLoad[8];
                                msg_test[1]=rxFrame.PayLoad[9];
                                msg_test[2]=rxFrame.PayLoad[10];
                                if (atoi(msg_test)==serial_number_n)
                                {
                                        modo="rede";
                                }
                }
        }
}

//se acaba de receber o codigo de modo normal
if ((rxFrame.PayLoad[0]=='!') && (rxFrame.PayLoad[1]=='N'))
{
        //verifica se é o mesmo numero de ano
        msg_test[0]=rxFrame.PayLoad[2];
        msg_test[1]=rxFrame.PayLoad[3];
        msg_test[2]=rxFrame.PayLoad[4];
        if (atoi(msg_test)==serial_number_a)
        {
```

```
                msg_test[0]=rxFrame.PayLoad[5];
                msg_test[1]=rxFrame.PayLoad[6];
                msg_test[2]=rxFrame.PayLoad[7];
                //verifica se é o mesmo numero de semana
                if (atoi(msg_test)==serial_number_s)
                {
                        msg_test[0]=rxFrame.PayLoad[8];
                        msg_test[1]=rxFrame.PayLoad[9];
                        msg_test[2]=rxFrame.PayLoad[10];
                        if (atoi(msg_test)==serial_number_n)
                        {
                                modo="normal";
                        }
                }
        }
}

//se acaba de receber o codigo de escrita
if ((rxFrame.PayLoad[0]=='$') && (rxFrame.PayLoad[1]=='#'))
{
        //o primeiro valor é o da do ano do serial_number_a
        msg_test[0]=rxFrame.PayLoad[2];
        msg_test[1]=rxFrame.PayLoad[3];
        msg_test[2]=rxFrame.PayLoad[4];
        Grava_EEP(EEP_SN_A,atoi(msg_test));



        //passa o X à frente e passa para o serial_number_s
        msg_test[0]=rxFrame.PayLoad[5];
        msg_test[1]=rxFrame.PayLoad[6];
        msg_test[2]=rxFrame.PayLoad[7];
        Grava_EEP(EEP_SN_S,atoi(msg_test));



        //passa o X à frente e passa para o serial_number_n
        msg_test[0]=rxFrame.PayLoad[8];
        msg_test[1]=rxFrame.PayLoad[9];
        msg_test[2]=rxFrame.PayLoad[10];
        Grava_EEP(EEP_SN_N,atoi(msg_test));



        //passa o X à frente e passa para a fraccao
        msg_test[0]=rxFrame.PayLoad[11];
        msg_test[1]=rxFrame.PayLoad[12];
        msg_test[2]=rxFrame.PayLoad[13];
        Grava_EEP(EEP_FRACAO,atoi(msg_test));

        //passa o X à frente e passa para a powerdown_h
        msg_test[0]=rxFrame.PayLoad[14];
        msg_test[1]=rxFrame.PayLoad[15];
        msg_test[2]=rxFrame.PayLoad[16];
        Grava_EEP(EEP_PWRDWNH,atoi(msg_test));


        //passa o X à frente e passa para a powerdown_l
        msg_test[0]=rxFrame.PayLoad[17];
        msg_test[1]=rxFrame.PayLoad[18];
        msg_test[2]=rxFrame.PayLoad[19];
        Grava_EEP(EEP_PWRDWNL,atoi(msg_test));

        //passa o X à frente e passa para a tarifa01_h
        msg_test[0]=rxFrame.PayLoad[20];
        msg_test[1]=rxFrame.PayLoad[21];
        msg_test[2]=rxFrame.PayLoad[22];
        Grava_EEP(EEP_TAR01H,atoi(msg_test));

        //passa o X à frente e passa para a tarifa01_l
        msg_test[0]=rxFrame.PayLoad[23];
        msg_test[1]=rxFrame.PayLoad[24];
        msg_test[2]=rxFrame.PayLoad[25];
        Grava_EEP(EEP_TAR01L,atoi(msg_test));
```

```
                                        //passa o X à frente e passa para a total_facturado_h
                                        msg_test[0]=rxFrame.PayLoad[26];
                                        msg_test[1]=rxFrame.PayLoad[27];
                                        msg_test[2]=rxFrame.PayLoad[28];
                                        Grava_EEP(EEP_TOTALH,atoi(msg_test));

                                        //passa o X à frente e passa para a total_facturado_m
                                        msg_test[0]=rxFrame.PayLoad[29];
                                        msg_test[1]=rxFrame.PayLoad[30];
                                        msg_test[2]=rxFrame.PayLoad[31];
                                        Grava_EEP(EEP_TOTALM,atoi(msg_test));


                                        //passa o X à frente e passa para a total_facturado_l
                                        msg_test[0]=rxFrame.PayLoad[32];
                                        msg_test[1]=rxFrame.PayLoad[33];
                                        msg_test[2]=rxFrame.PayLoad[34];
                                        Grava_EEP(EEP_TOTALL,atoi(msg_test));


                                        //passa o X à frente e passa para a password_h
                                        msg_test[0]=rxFrame.PayLoad[35];
                                        msg_test[1]=rxFrame.PayLoad[36];
                                        msg_test[2]=rxFrame.PayLoad[37];
                                        Grava_EEP(EEP_PASSWM,atoi(msg_test));



                                        //passa o X à frente e passa para a password_m
                                        msg_test[0]=rxFrame.PayLoad[38];
                                        msg_test[1]=rxFrame.PayLoad[39];
                                        msg_test[2]=rxFrame.PayLoad[40];
                                        Grava_EEP(EEP_PASSWM,atoi(msg_test));



                                        //passa o X à frente e passa para a password_l
                                        msg_test[0]=rxFrame.PayLoad[41];
                                        msg_test[1]=rxFrame.PayLoad[42];
                                        msg_test[2]=rxFrame.PayLoad[43];
                                        Grava_EEP(EEP_PASSWL,atoi(msg_test));
                                }
                                DiscardPacket();
                                // Toggle LED2 to indicate receiving a packet.
        LED_2_W=~LED_2_W;
      }
    }
}

//---------------------------------------------------------------------------
// High priority interrupt routine

#pragma code

#pragma interrupt LInterruptHandle
void LInterruptHandle ()
{
            int i;

            //sempre que há uma interrupção do timer da contagem
            if (PIR1bits.TMR2IF)
   {
                        if (ciclo_actual>=n_ciclos_contagem_f)
                        {
                                    val_pagar=val_pagar+fraccao;
                                    //ajusta as variaveis de cada display para mostrar
                                    valor_a_pagar_d_5=val_pagar/10000;
                                    valor_a_pagar_d_4=(val_pagar-(valor_a_pagar_d_5*10000))/1000;
                                    valor_a_pagar_d_3=(val_pagar-
(valor_a_pagar_d_5*10000+valor_a_pagar_d_4*1000))/100;
                                    valor_a_pagar_d_2=(val_pagar-
(valor_a_pagar_d_5*10000+valor_a_pagar_d_4*1000+valor_a_pagar_d_3*100))/10;
                                    valor_a_pagar_d_1=val_pagar-
(valor_a_pagar_d_5*10000+valor_a_pagar_d_4*1000+valor_a_pagar_d_3*100+valor_a_pagar_d_2*10);
```

```
                        //lê e actualiza o total facturado
                        total_facturado=total_facturado+fraccao;
                //
        total_facturado=(65536*LER_EEP(EEP_TOTALH))+(256*LER_EEP(EEP_TOTALM))+LER_EEP(EEP_TOTALL) +
fraccao;

                        //total_facturado=1193046;

                        total_facturado_h=total_facturado/65536;
                        total_facturado_m=(total_facturado-(total_facturado_h*65536))/256;
                        total_facturado_l=(total_facturado-(total_facturado_h*65536))-
(total_facturado_m*256);

                        if (modo=="rede")
                        {

                                //prepara a string
                                strcpy(msg_enviar2,msgci2);             //mete o código inicial2
                                strcat(msg_enviar2,msgsn);              //mete o SN -> serial number
                                        itoa((serial_number_a),info);
                                        strcat(msg_enviar2,info);
                                strcat(msg_enviar2,msgx);               //separa com X
                                        itoa((serial_number_s),info);
                                        strcat(msg_enviar2,info);
                                strcat(msg_enviar2,msgx);               //separa com X
                                        itoa((serial_number_n),info);
                                        strcat(msg_enviar2,info);
                                strcat(msg_enviar2,msgx);               //separa com X
                                        itoa((val_pagar),info);                 //a pagar:
                                        strcat(msg_enviar2,info);
                                strcat(msg_enviar2,msgcf);              //mete o codigo final

                                //envia a string
                                FlushTx();
                                for(i = 0; i < (strlen(msg_enviar2)); i++)
                                {
                                        WriteData(msg_enviar2[i]);
                                }

                                //UnicastLongAddress(0,1122334455667703, FALSE, FALSE);
                                UnicastConnection(0, FALSE, TRUE);
                                //BroadcastPacket(myPANID, FALSE, FALSE);

                        }

                        Delay100TCYx(120);
                        Grava_EEP(EEP_TOTALM,total_facturado_m);
                        Delay100TCYx(120);
                        Grava_EEP(EEP_TOTALL,total_facturado_l);
                        Delay100TCYx(120);
                        Grava_EEP(EEP_TOTALH,total_facturado_h);
                        Delay100TCYx(120);
                        ciclo_actual=0;
                }
                else
                {
                        ciclo_actual=ciclo_actual+1;
                }
                //limpa a flag
                PIR1bits.TMR2IF=0;
 }

        //interrupção do timer3 que está sempre activo
        if (PIR2bits.TMR3IF)
        {
                if (gaveta=="just_closed")
                {
                        //se já passaram 5seg, muda o nome da variavel gaveta
                        if (ciclo_3>=38)
                        {
                                gaveta="fechada";
                                ciclo_3=0;

                                //caso esteja em rede, avisa que acabou de fechar o jogo
                                if (modo=="rede")
```

```
                                        {

                                                //prepara a string
                                                strcpy(msg_enviar2,msgci2);             //mete o código inicial2
                                                strcat(msg_enviar2,msgsn);              //mete o SN -> serial
number
                                                        itoa((serial_number_a),info);
                                                        strcat(msg_enviar2,info);
                                                strcat(msg_enviar2,msgx);               //separa com X
                                                        itoa((serial_number_s),info);
                                                        strcat(msg_enviar2,info);
                                                strcat(msg_enviar2,msgx);               //separa com X
                                                        itoa((serial_number_n),info);
                                                        strcat(msg_enviar2,info);
                                                strcat(msg_enviar2,msgx);               //separa com X
                                                        strcat(msg_enviar2,msgfg);      //mete o caracter FG
                                                strcat(msg_enviar2,msgcf);              //mete o codigo final

                                                //envia a string
                                                FlushTx();
                                                for(i = 0; i < (strlen(msg_enviar2)); i++)
                                                {
                                                        WriteData(msg_enviar2[i]);
                                                }

                                                //UnicastLongAddress(0,1122334455667703, FALSE, FALSE);
                                                UnicastConnection(0, FALSE, TRUE);
                                                //BroadcastPacket(myPANID, FALSE, FALSE);

                                        }
                                }
                                else
                                {
                                        ciclo_3=ciclo_3+1;
                                }
                        }

                        if (gaveta=="faz_nada")
                        {
                                if (ciclo_4>=1)
                                {
                                        //alterna entre 0 e 1 o estado do display
                                        if (estado_displays2==0)
                                        {
                                                estado_displays2=1;
                                        }
                                        else
                                        {
                                                estado_displays2=0;
                                        }
                                        ciclo_4=0;
                                }
                                else
                                {
                                        ciclo_4=ciclo_4+1;
                                }
                        }

                        if (gaveta=="alarme")
                        {
                                //espera 2seg (para ser precisao são 38 ciclos para 5seg)
                                if (ciclo_2>=3)
                                {
                                        //alterna entre 0 e 1 o estado do display
                                        if (estado_displays==0)
                                        {
                                                estado_displays=1;
                                        }
                                        else
                                        {
                                                estado_displays=0;
                                        }
                                        ciclo_2=0;
                                }
                                else
```

```
                {
                        ciclo_2=ciclo_2+1;
                }
        }

//      LED_2_W=~LED_2_W;
        PIR2bits.TMR3IF=0;
    }
}


//grava na eeprom
void Grava_EEP(unsigned int eep_endereco, unsigned int eep_valor)
{
    EEADR =eep_endereco;
    EEDATA = eep_valor;
    EECON1bits.EEPGD =0;
    EECON1bits.CFGS =0;
    EECON1bits.WREN =1;
    INTCONbits.GIE = 0;
    EECON2 = 0x55;
    EECON2 = 0xAA;
    EECON1bits.WR = 1;
    INTCONbits.GIE = 1;
    EECON1bits.WREN = 0;
    Delay100TCYx(155);
    Delay100TCYx(155);
    Delay100TCYx(155);
}

//lê a eeprom
unsigned int LER_EEP(unsigned int eep_endereco)
{
    unsigned int valor_lido;
    EEADR =eep_endereco;
    EECON1bits.EEPGD = 0; // address eeprom, not program
    EECON1bits.CFGS =0;
    EECON1bits.RD = 1;
    valor_lido=EEDATA;
    Delay100TCYx(155);
    return valor_lido;
}


//lê a posicão de memória e calcula a tarifa
unsigned int Calculo_Tarifa(unsigned int num_tarifa)
{
    unsigned int tarifa_usar;

    if (num_tarifa==1)
    {
        unsigned int tarifaa=0;

        //lê a eeprom
        tarifa_usar=256*LER_EEP(EEP_TAR01H)+LER_EEP(EEP_TAR01L);

        tarifa_d_5=tarifa_usar/10000;
        tarifa_d_4=(tarifa_usar-(tarifa_d_5*10000))/1000;
        tarifa_d_3=(tarifa_usar-(tarifa_d_5*10000+tarifa_d_4*1000))/100;
        tarifa_d_2=(tarifa_usar-(tarifa_d_5*10000+tarifa_d_4*1000+tarifa_d_3*100))/10;
        tarifa_d_1=(tarifa_usar-(tarifa_d_5*10000+tarifa_d_4*1000+tarifa_d_3*100+tarifa_d_2*10));
    }

    return tarifa_usar;
}

//lê a posicao de memoria relativa a fracao
unsigned int Calculo_Fraccao()
{
    unsigned int fraccao_usar;

    fraccao_usar=LER_EEP(EEP_FRACAO);

    return fraccao_usar;
}
```

```
//lê a posicao de memoria relativa a fracao
unsigned char verifica_sensores()
{
	int i;

	unsigned char estado_sensores;

		if ((SENSOR_1) && (SENSOR_2) && (SENSOR_3) && (SENSOR_4))
		{
			estado_sensores='a';
		}
		else
		{
			if ((SENSOR_1==0) && (SENSOR_2==0) && (SENSOR_3==0) && (SENSOR_4)==0)
			{
				estado_sensores='f';
			}
			else
			{
				estado_sensores='e';
			}
		}

	//se a gaveta está aberta
		if (estado_sensores=='a')	//a-> aberto   e-> erro   f-> fechado
		{
			//vai abrir o jogo
			if (gaveta=="fechada")
			{
				val_pagar=fraccao;

				//define os valores dos displays
				valor_a_pagar_d_5=0;
				valor_a_pagar_d_4=0;
				valor_a_pagar_d_3=0;
				valor_a_pagar_d_2=fraccao/10;
				valor_a_pagar_d_1=fraccao-valor_a_pagar_d_2*10;

				//activa o jogo
				gaveta="em_jogo";
				ciclo_actual=0;


				//caso esteja em rede, manda o impulso inicial
				if (modo=="rede")
				{

					//prepara a string
					strcpy(msg_enviar2,msgci2);		//mete o código inicial2
					strcat(msg_enviar2,msgsn);		//mete o SN -> serial
number
						itoa((serial_number_a),info);
						strcat(msg_enviar2,info);
					strcat(msg_enviar2,msgx);		//separa com X
						itoa((serial_number_s),info);
						strcat(msg_enviar2,info);
					strcat(msg_enviar2,msgx);		//separa com X
						itoa((serial_number_n),info);
						strcat(msg_enviar2,info);
					strcat(msg_enviar2,msgx);		//separa com X
						itoa((val_pagar),info);			//a pagar:
						strcat(msg_enviar2,info);
					strcat(msg_enviar2,msgcf);		//mete o codigo final

					//envia a string
					FlushTx();
					for(i = 0; i < (strlen(msg_enviar2)); i++)
					{
						WriteData(msg_enviar2[i]);
					}

					//UnicastLongAddress(0,1122334455667703, FALSE, FALSE);
					UnicastConnection(0, FALSE, TRUE);
					//BroadcastPacket(myPANID, FALSE, FALSE);
```

```
                        }


                }

                //continua a contagem
                if (gaveta=="em_jogo")
                {

                }

                //se estava em alarme e voltou a estar em jogo
                if (gaveta=="alarme")
                {
                        gaveta="em_jogo";
                }

                //se estava quase a fechar e voltou a abrir
                if (gaveta=="just_closed")
                {
                        gaveta="em_jogo";
                }

                //comum
                mostra_tarifa(tarifa);
                valor_a_pagar();
        }

        //se a gaveta está em erro
        else if (estado_sensores=='e')  //a-> aberto   e-> erro   f-> fechado
        {
                //se estava em jogo e fica em alarme
                if (gaveta=="em_jogo")
                {
                        mostra_tarifa(tarifa);
                        valor_a_pagar(val_pagar);
                        gaveta="alarme";
                        estado_displays=0;
                }
                //se estava em alarme e continua em alarme
                if (gaveta=="alarme")
                {
                        if (estado_displays==0)
                        {
                                gaveta="alarme";
                        }
                        else
                        {
                                mostra_tarifa(tarifa);
                                valor_a_pagar(val_pagar);
                                gaveta="alarme";
                        }
                }
        }

        //se a gaveta está fechada
        else if (estado_sensores=='f')  //a-> aberto   e-> erro   f-> fechado
        {
                //se a gaveta estiver fechada e continuar fechada
                if (gaveta=="fechada")
                {
                        if ((PUSH_BUTTON_1==1) && (PUSH_BUTTON_2==0) &&
(PUSH_BUTTON_3==0))
                        {
                                Delay100TCYx(255);
                                gaveta="faz_nada";
                                programa_tarifa();
                        }
                        else
                        {
                                gaveta=="fechada";
                        }
                }
```

```
                        if (gaveta=="em_jogo")
                        {
                                mostra_tarifa(tarifa);
                                valor_a_pagar(val_pagar);
                                gaveta="just_closed";
                        }
                        if (gaveta=="alarme")
                        {
                                mostra_tarifa(tarifa);
                                valor_a_pagar(val_pagar);
                                gaveta="just_closed";
                        }
                        if (gaveta=="just_closed")
                        {
                                mostra_tarifa(tarifa);
                                valor_a_pagar(val_pagar);
                                //gaveta="fechada";
                        }
                }
}

//mostra a tarifa
unsigned int mostra_tarifa()
{
        //se o disp5=0
        if (tarifa_d_5==0)
        {
                //se o disp4=0
                if (tarifa_d_4==0)
                {
                }
                else
                {
                        disp_a_mostrar=4;
                        display_numero(valor_a_pagar_d_4);
                        activa_display(disp_a_mostrar);
                        Delay100TCYx(120);
                }
        }
        else
        {
                disp_a_mostrar=5;
                display_numero(tarifa_d_5);
                activa_display(disp_a_mostrar);
                Delay100TCYx(120);

                disp_a_mostrar=4;
                display_numero(tarifa_d_4);
                activa_display(disp_a_mostrar);
                Delay100TCYx(120);
        }

        disp_a_mostrar=3;
        display_numero_virgula(tarifa_d_3);
        activa_display(disp_a_mostrar);
        Delay100TCYx(120);

        disp_a_mostrar=2;
        display_numero(tarifa_d_2);
        activa_display(disp_a_mostrar);
        Delay100TCYx(120);

        disp_a_mostrar=1;
        display_numero(tarifa_d_1);
        activa_display(disp_a_mostrar);
        Delay100TCYx(120);

        disp_a_mostrar=0;
        activa_display(disp_a_mostrar);
}


//menu de programação de tarifa
unsigned int programa_tarifa()
{
```

```
	unsigned int
psswrd,val_a_mostrar,disp_a_acender,digito01,digito02,digito03,digito04,digito05,disp_a_acender2;
	val_a_mostrar=0;
	disp_a_acender=5;
	digito01=0;
	digito02=0;
	digito03=0;
	digito04=0;
	digito05=0;

	//enquanto não carregar no OK
	while (PUSH_BUTTON_1==1)
	{

		//se carregou em incrementar
		if (PUSH_BUTTON_2==0)
		{
			Delay10KTCYx(50);
			if (val_a_mostrar==9)
			{
				val_a_mostrar=0;
			}
			else
			{
				val_a_mostrar=val_a_mostrar+1;
			}
			if (disp_a_acender==1)
			{
				digito01=val_a_mostrar;
			}
			else if (disp_a_acender==2)
			{
				digito02=val_a_mostrar;
			}
			else if (disp_a_acender==3)
			{
				digito03=val_a_mostrar;
			}
			else if (disp_a_acender==4)
			{
				digito04=val_a_mostrar;
			}
			else if (disp_a_acender==5)
			{
				digito05=val_a_mostrar;
			}

		}

		//se carregou em cursor
		if (PUSH_BUTTON_3==0)
		{
			Delay10KTCYx(50);
			if (disp_a_acender==1)
			{
				disp_a_acender=5;
			}
			else
			{
				disp_a_acender=disp_a_acender-1;
			}
		}
		if ((disp_a_acender==5) && estado_displays2==1)
		{
		}
		else
		{
			display_numero(digito05);
			disp_a_acender2=5;
			activa_display(disp_a_acender2);
			Delay100TCYx(150);
		}

		if ((disp_a_acender==4) && estado_displays2==1)
		{
```

```
		}
		else
		{
			display_numero(digito04);
			disp_a_acender2=4;
			activa_display(disp_a_acender2);
			Delay100TCYx(150);
		}
		if ((disp_a_acender==3) && estado_displays2==1)
		{
		}
		else
		{
			display_numero(digito03);
			disp_a_acender2=3;
			activa_display(disp_a_acender2);
			Delay100TCYx(150);
		}
		if ((disp_a_acender==2) && estado_displays2==1)
		{
		}
		else
		{
			display_numero(digito02);
			disp_a_acender2=2;
			activa_display(disp_a_acender2);
			Delay100TCYx(150);
		}
		if ((disp_a_acender==1) && estado_displays2==1)
		{
		}
		else
		{
			display_numero(digito01);
			disp_a_acender2=1;
			activa_display(disp_a_acender2);
			Delay100TCYx(150);
		}
}
//espera que deslargue o butão
while (PUSH_BUTTON_1==0)
{
}

//se a password for correcta
if (digito05*10000+digito04*1000+digito03*100+digito02*10+digito01==password)
{
		val_a_mostrar=0;
		disp_a_acender2=5;
		digito01=tarifa_d_1;
		digito02=tarifa_d_2;
		digito03=tarifa_d_3;
		digito04=0;
		digito05=0;



		//enquanto não carregar no OK
		while (PUSH_BUTTON_1==1)
		{

			//se carregou em incrementar
			if (PUSH_BUTTON_2==0)
			{
				Delay10KTCYx(50);

				if (disp_a_acender==1)
				{
					if (digito01==9)
					{
						digito01==0;
					}
					else
					{
```

```
                                digito01=digito01+1;
                        }
                }
                else if (disp_a_acender==2)
                {
                        if (digito02==9)
                        {
                                digito02==0;
                        }
                        else
                        {
                                digito02=digito02+1;
                        }
                }
                else if (disp_a_acender==3)
                {
                        if (digito03==9)
                        {
                                digito03==0;
                        }
                        else
                        {
                                digito03=digito03+1;
                        }
                }
                else if (disp_a_acender==4)
                {
                        if (digito04==9)
                        {
                                digito04==0;
                        }
                        else
                        {
                                digito04=digito04+1;
                        }
                }
                else if (disp_a_acender==5)
                {
                        if (digito05==9)
                        {
                                digito05==0;
                        }
                        else
                        {
                                digito05=digito05+1;
                        }
                }

        }

        //se carregou em cursor
        if (PUSH_BUTTON_3==0)
        {
                Delay10KTCYx(50);
                if (disp_a_acender==1)
                {
                        disp_a_acender=5;
                }
                else
                {
                        disp_a_acender=disp_a_acender-1;
                }
        }

        //vai piscar o que está seleccionado
        if ((disp_a_acender==5) && estado_displays2==1)
        {
        }
        else
        {
                display_numero(digito05);
                disp_a_acender2=5;
                activa_display(disp_a_acender2);
                Delay100TCYx(150);
        }
```

```
                        if ((disp_a_acender==4) && estado_displays2==1)
                        {
                        }
                        else
                        {
                                display_numero(digito04);
                                disp_a_acender2=4;
                                activa_display(disp_a_acender2);
                                Delay100TCYx(150);
                        }
                        if ((disp_a_acender==3) && estado_displays2==1)
                        {
                        }
                        else
                        {
                                display_numero(digito03);
                                disp_a_acender2=3;
                                activa_display(disp_a_acender2);
                                Delay100TCYx(150);
                        }
                        if ((disp_a_acender==2) && estado_displays2==1)
                        {
                        }
                        else
                        {
                                display_numero(digito02);
                                disp_a_acender2=2;
                                activa_display(disp_a_acender2);
                                Delay100TCYx(150);
                        }
                        if ((disp_a_acender==1) && estado_displays2==1)
                        {
                        }
                        else
                        {
                                display_numero(digito01);
                                disp_a_acender2=1;
                                activa_display(disp_a_acender2);
                                Delay100TCYx(150);
                        }

                }
                //espera que largue o butão OK
                while (PUSH_BUTTON_1==0)
                {
                }

                tarifa=digito05*10000+digito04*1000+digito03*100+digito04*10+digito01;
                tarifa_save_h=tarifa/256;
                tarifa_save_l=tarifa-tarifa_save_h*256;

                Grava_EEP(EEP_TAR01H,tarifa_save_h);
                Delay100TCYx(155);
                Grava_EEP(EEP_TAR01L,tarifa_save_l);
                Delay100TCYx(155);

                tarifa=Calculo_Tarifa(1);


        }
        gaveta="fechada";
        disp_a_acender=0;
        activa_display(disp_a_acender);
}



//mostra o total a pagar
unsigned int valor_a_pagar()
{
        //se o disp5=0
        if (valor_a_pagar_d_5==0)
        {
                //se o disp4=0
```

```
                if (valor_a_pagar_d_4==0)
                {
                }
                else
                {
                        disp_a_mostrar=9;
                        display_numero(valor_a_pagar_d_4);
                        activa_display(disp_a_mostrar);
                        Delay100TCYx(120);
                }
        }
        else
        {
                disp_a_mostrar=10;
                display_numero(valor_a_pagar_d_5);
                activa_display(disp_a_mostrar);
                Delay100TCYx(120);

                disp_a_mostrar=9;
                display_numero(valor_a_pagar_d_4);
                activa_display(disp_a_mostrar);
                Delay100TCYx(120);
        }


        disp_a_mostrar=8;
        display_numero_virgula(valor_a_pagar_d_3);
        activa_display(disp_a_mostrar);
        Delay100TCYx(120);

        disp_a_mostrar=7;
        display_numero(valor_a_pagar_d_2);
        activa_display(disp_a_mostrar);
        Delay100TCYx(120);

        disp_a_mostrar=6;
        display_numero(valor_a_pagar_d_1);
        activa_display(disp_a_mostrar);
        Delay100TCYx(120);

        disp_a_mostrar=0;
        activa_display(disp_a_mostrar);
}
```

SymbolTime.c

```
#include "SystemProfile.h"
#include "SymbolTime.h"
#include "Compiler.h"
#include "GenericTypeDefs.h"

volatile BYTE timerExtension1,timerExtension2;

//This function will configure the UART for use at in 8 bits, 1 stop, no flowcontrol mode
void InitSymbolTimer()
{
   T0CON = 0b00000000 | CLOCK_DIVIDER_SETTING;
   INTCON2bits.TMR0IP = 1;
   INTCONbits.TMR0IF = 0;
   INTCONbits.TMR0IE = 1;
   T0CONbits.TMR0ON = 1;

   timerExtension1 = 0;
   timerExtension2 = 0;
}

//This function returns the current time
TICK TickGet(void)
{
   TICK currentTime;
   /* disable the timer to prevent roll over of the lower 16 bits while before/after reading of the extension */
   INTCONbits.TMR0IE = 0;

   /* copy the byte extension */
   currentTime.byte.b2 = 0;
   currentTime.byte.b3 = 0;

   /* read the timer value */
   currentTime.byte.b0 = TMR0L;
   currentTime.byte.b1 = TMR0H;

   //if an interrupt occured after IE = 0, then we need to figure out if it was
   //before or after we read TMR0L
   if(INTCONbits.TMR0IF)
   {
      if(currentTime.byte.b0<10)
      {
         //if we read TMR0L after the rollover that caused the interrupt flag then we need
         //to increment the 3rd byte
         currentTime.byte.b2++;  //increment the upper most
         if(timerExtension1 == 0xFF)
         {
            currentTime.byte.b3++;
         }
      }
   }

   /* copy the byte extension */
   currentTime.byte.b2 += timerExtension1;
   currentTime.byte.b3 += timerExtension2;

   /* enable the timer*/
   INTCONbits.TMR0IE = 1;
}
```

Displays.c (Contador de Tempo de Bilhar)

```
//codigo dos displays
#include "Console.h"
#include "P2P.h"
#include "MRF24J40.h"
#include "SymbolTime.h"
void activa_display(unsigned int n_disp)
{
	if (n_disp==0)		//se está tudo apagado
	{
		DA0=1;
		DA1=1;
		DA2=1;
		DA3=1;
	}
	else if (n_disp==1)	//se liga o 1º display (cima direita)
	{
		DA0=0;
		DA1=0;
		DA2=0;
		DA3=0;
	}
	else if (n_disp==2)
	{
		DA0=1;
		DA1=0;
		DA2=0;
		DA3=0;
	}
	else if (n_disp==3)
	{
		DA0=0;
		DA1=1;
		DA2=0;
		DA3=0;
	}
	else if (n_disp==4)
	{
		DA0=1;
		DA1=1;
		DA2=0;
		DA3=0;
	}
	else if (n_disp==5)
	{
		DA0=0;
		DA1=0;
		DA2=1;
		DA3=0;
	}
	else if (n_disp==6)	//baixo direita
	{
		DA0=1;
		DA1=0;
		DA2=0;
		DA3=1;
	}
	else if (n_disp==7)
	{
		DA0=0;
		DA1=0;
		DA2=0;
		DA3=1;
	}
	else if (n_disp==8)
	{
		DA0=1;
		DA1=1;
```

```
                DA2=1;
                DA3=0;
        }
        else if (n_disp==9)
        {
                DA0=0;
                DA1=1;
                DA2=1;
                DA3=0;
        }
        else if (n_disp==10)
        {
                DA0=1;
                DA1=0;
                DA2=1;
                DA3=0;
        }
        else
        {
                DA0=1;
                DA1=1;
                DA2=1;
                DA3=1;
        }
}

void display_numero(int num1)
{
        if (num1==0)
        {
                DS1=1;    //baixo esquerda
                DS2=1;    //baixo
                DS3=0;    //ponto
                DS4=1;    //baixo direita
                DS5=1;    //cima direita
                DS6=1;    //cima
                DS7=1;    //cima esquerda
                DS8=0;    //meio
        }
        else if (num1==1)
        {
                DS1=0;
                DS2=0;
                DS3=0;
                DS4=1;
                DS5=1;
                DS6=0;
                DS7=0;
                DS8=0;
        }
        else if (num1==2)
        {
                DS1=1;
                DS2=1;
                DS3=0;
                DS4=0;
                DS5=1;
                DS6=1;
                DS7=0;
                DS8=1;
        }
        else if (num1==3)
        {
                DS1=0;
                DS2=1;
                DS3=0;
                DS4=1;
                DS5=1;
                DS6=1;
                DS7=0;
                DS8=1;
        }
        else if (num1==4)
        {
```

```
                DS1=0;
                DS2=0;
                DS3=0;
                DS4=1;
                DS5=1;
                DS6=0;
                DS7=1;
                DS8=1;
        }
        else if (num1==5)
        {
                DS1=0;
                DS2=1;
                DS3=0;
                DS4=1;
                DS5=0;
                DS6=1;
                DS7=1;
                DS8=1;
        }
        else if (num1==6)
        {
                DS1=1;
                DS2=1;
                DS3=0;
                DS4=1;
                DS5=0;
                DS6=1;
                DS7=1;
                DS8=1;
        }
        else if (num1==7)
        {
                DS1=0;
                DS2=0;
                DS3=0;
                DS4=1;
                DS5=1;
                DS6=1;
                DS7=0;
                DS8=0;
        }
        else if (num1==8)
        {
                DS1=1;
                DS2=1;
                DS3=0;
                DS4=1;
                DS5=1;
                DS6=1;
                DS7=1;
                DS8=1;
        }
        else if (num1==9)
        {
                DS1=0;
                DS2=1;
                DS3=0;
                DS4=1;
                DS5=1;
                DS6=1;
                DS7=1;
                DS8=1;
        }
}

void display_numero_virgula(int num1)
{
        if (num1==0)
        {
                DS1=1;    //baixo esquerda
                DS2=1;    //baixo
                DS3=1;    //ponto
                DS4=1;    //baixo direita
                DS5=1;    //cima direita
```

```
            DS6=1;     //cima
            DS7=1;     //cima esquerda
            DS8=0;     //meio
    }
    else if (num1==1)
    {
            DS1=0;
            DS2=0;
            DS3=1;
            DS4=1;
            DS5=1;
            DS6=0;
            DS7=0;
            DS8=0;
    }
    else if (num1==2)
    {
            DS1=1;
            DS2=1;
            DS3=1;
            DS4=0;
            DS5=1;
            DS6=1;
            DS7=0;
            DS8=1;
    }
    else if (num1==3)
    {
            DS1=0;
            DS2=1;
            DS3=1;
            DS4=1;
            DS5=1;
            DS6=1;
            DS7=0;
            DS8=1;
    }
    else if (num1==4)
    {
            DS1=0;
            DS2=0;
            DS3=1;
            DS4=1;
            DS5=1;
            DS6=0;
            DS7=1;
            DS8=1;
    }
    else if (num1==5)
    {
            DS1=0;
            DS2=1;
            DS3=1;
            DS4=1;
            DS5=0;
            DS6=1;
            DS7=1;
            DS8=1;
    }
    else if (num1==6)
    {
            DS1=1;
            DS2=1;
            DS3=1;
            DS4=1;
            DS5=0;
            DS6=1;
            DS7=1;
            DS8=1;
    }
    else if (num1==7)
    {
            DS1=0;
            DS2=0;
            DS3=1;
```

```
                DS4=1;
                DS5=1;
                DS6=1;
                DS7=0;
                DS8=0;
        }
        else if (num1==8)
        {
                DS1=1;
                DS2=1;
                DS3=1;
                DS4=1;
                DS5=1;
                DS6=1;
                DS7=1;
                DS8=1;
        }
        else if (num1==9)
        {
                DS1=0;
                DS2=1;
                DS3=1;
                DS4=1;
                DS5=1;
                DS6=1;
                DS7=1;
                DS8=1;
        }
}

//LETRAS: Não existem: K - M - Q - X
void display_letra(unsigned char str)
{

    if (str=='A')
        {
                DS1=1;
                DS2=0;
                DS3=0;
                DS4=1;
                DS5=1;
                DS6=1;
                DS7=1;
                DS8=1;
        }
        else if (str=='B')
        {
                DS1=1;
                DS2=1;
                DS3=0;
                DS4=1;
                DS5=0;
                DS6=0;
                DS7=1;
                DS8=1;
        }
        else if (str=='C')
        {
                DS1=1;
                DS2=1;
                DS3=0;
                DS4=0;
                DS5=0;
                DS6=1;
                DS7=1;
                DS8=0;
        }
        else if (str=='D')
        {
                DS1=1;
                DS2=1;
                DS3=0;
                DS4=1;
                DS5=1;
                DS6=0;
```

```
		DS7=0;
		DS8=1;
	}
	else if (str=='E')
	{
		DS1=1;
		DS2=1;
		DS3=0;
		DS4=0;
		DS5=0;
		DS6=1;
		DS7=1;
		DS8=1;
	}
	else if (str=='F')
	{
		DS1=1;
		DS2=0;
		DS3=0;
		DS4=0;
		DS5=0;
		DS6=1;
		DS7=1;
		DS8=1;
	}
	else if (str=='G')
	{
		DS1=1;
		DS2=1;
		DS3=0;
		DS4=1;
		DS5=0;
		DS6=1;
		DS7=1;
		DS8=1;
	}
	else if (str=='H')
	{
		DS1=1;
		DS2=0;
		DS3=0;
		DS4=1;
		DS5=1;
		DS6=0;
		DS7=1;
		DS8=1;
	}
	else if (str=='I')
	{
		DS1=0;
		DS2=0;
		DS3=0;
		DS4=1;
		DS5=1;
		DS6=0;
		DS7=0;
		DS8=0;
	}
	else if (str=='J')
	{
		DS1=1;
		DS2=1;
		DS3=0;
		DS4=1;
		DS5=1;
		DS6=1;
		DS7=0;
		DS8=0;
	}
	else if (str=='L')
	{
		DS1=1;
		DS2=1;
		DS3=0;
		DS4=0;
```

```
		DS5=0;
		DS6=0;
		DS7=1;
		DS8=0;
	}
	else if (str=='N')
	{
		DS1=1;
		DS2=0;
		DS3=0;
		DS4=1;
		DS5=0;
		DS6=0;
		DS7=0;
		DS8=1;
	}
	else if (str=='O')
	{
		DS1=1;
		DS2=1;
		DS3=0;
		DS4=1;
		DS5=1;
		DS6=1;
		DS7=1;
		DS8=0;
	}
	else if (str=='P')
	{
		DS1=1;
		DS2=0;
		DS3=0;
		DS4=0;
		DS5=1;
		DS6=1;
		DS7=1;
		DS8=1;
	}
	else if (str=='R')
	{
		DS1=1;
		DS2=0;
		DS3=0;
		DS4=0;
		DS5=0;
		DS6=0;
		DS7=0;
		DS8=1;
	}
	else if (str=='S')
	{
		DS1=0;
		DS2=1;
		DS3=0;
		DS4=1;
		DS5=0;
		DS6=1;
		DS7=1;
		DS8=1;
	}
	else if (str=='T')
	{
		DS1=1;
		DS2=1;
		DS3=0;
		DS4=0;
		DS5=0;
		DS6=0;
		DS7=1;
		DS8=1;
	}
	else if (str=='U')
	{
		DS1=1;
		DS2=1;
```

```
                DS3=0;
                DS4=1;
                DS5=1;
                DS6=0;
                DS7=1;
                DS8=0;
        }
        else if (str=='V')
        {
                DS1=1;
                DS2=1;
                DS3=0;
                DS4=1;
                DS5=0;
                DS6=0;
                DS7=0;
                DS8=0;
        }
        else if (str=='Z')
        {
                DS1=1;
                DS2=1;
                DS3=0;
                DS4=0;
                DS5=1;
                DS6=1;
                DS7=0;
                DS8=1;
        }

}
```

Compiler.h

```
#ifndef __COMPILER_H
#define __COMPILER_H
#include <p18cxxx.h>
#define INSTR_FREQ                      (CLOCK_FREQ/4)
#include "HardwareProfile.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define ROM                     rom
#define strcpypgm2ram(a, b)             strcpypgm2ram(a,(far rom char*)b)
#define   __attribute__(a)
```

Console.h

```
#ifndef  _CONSOLE_H_
#define  _CONSOLE_H_
#include "GenericTypeDefs.h"
#include "Compiler.h"
#include "SystemProfile.h"
   #if defined(ENABLE_CONSOLE)    // Useful for disabling the console (saving power)
      void ConsoleInit(void);
      #define ConsoleIsPutReady()     (TXSTAbits.TRMT)
      void ConsolePut(BYTE c);
      void ConsolePutString(BYTE *s);
      void ConsolePutROMString(ROM char* str);

      #define ConsoleIsGetReady()     (PIR1bits.RCIF)
      BYTE ConsoleGet(void);
      BYTE ConsoleGetString(char *buffer, BYTE bufferLen);
      void PrintChar(BYTE);
      void PrintDec(BYTE);
   #else
      #define ConsoleInit()
      #define ConsoleIsPutReady() 1
      #define ConsolePut(c)
      #define ConsolePutString(s)
      #define ConsolePutROMString(str)

      #define ConsoleIsGetReady() 1
      #define ConsoleGet()        'a'
      #define ConsoleGetString(buffer, bufferLen) 0
      #define PrintChar(a)
      #define PrintDec(a)
   #endif
#define printf(x) ConsolePutROMString((ROM char*)x)
```

GenericTypeDef.h

```
#ifndef __GENERIC_TYPE_DEFS_H_
#define __GENERIC_TYPE_DEFS_H_
typedef enum _BOOL { FALSE = 0, TRUE } BOOL;    // Undefined size
typedef unsigned char                   BYTE;                   // 8-bit unsigned
typedef unsigned short int      WORD;                   // 16-bit unsigned
typedef unsigned long                   DWORD;                  // 32-bit unsigned
typedef signed char                     CHAR;                   // 8-bit signed
typedef signed short int        SHORT;                  // 16-bit signed
typedef signed long                     LONG;                   // 32-bit signed
typedef union _BYTE_VAL
{
   BYTE Val;
   struct
   {
      unsigned char b0:1;
      unsigned char b1:1;
      unsigned char b2:1;
      unsigned char b3:1;
      unsigned char b4:1;
      unsigned char b5:1;
      unsigned char b6:1;
      unsigned char b7:1;
   } bits;
} BYTE_VAL;
typedef union _WORD_VAL
{
   WORD Val;
   BYTE v[2];
   struct
   {
      BYTE LB;
      BYTE HB;
   } byte;
   struct
   {
      unsigned char b0:1;
      unsigned char b1:1;
      unsigned char b2:1;
      unsigned char b3:1;
      unsigned char b4:1;
      unsigned char b5:1;
      unsigned char b6:1;
      unsigned char b7:1;
      unsigned char b8:1;
      unsigned char b9:1;
      unsigned char b10:1;
      unsigned char b11:1;
      unsigned char b12:1;
      unsigned char b13:1;
      unsigned char b14:1;
      unsigned char b15:1;
   } bits;
} WORD_VAL;
typedef union _DWORD_VAL
{
   DWORD Val;
         WORD w[2];
   BYTE v[4];
   struct
   {
      WORD LW;
      WORD HW;
   } word;
   struct
   {
      BYTE LB;
      BYTE HB;
      BYTE UB;
```

```
        BYTE MB;
    } byte;
    struct
    {
        unsigned char b0:1;
        unsigned char b1:1;
        unsigned char b2:1;
        unsigned char b3:1;
        unsigned char b4:1;
        unsigned char b5:1;
        unsigned char b6:1;
        unsigned char b7:1;
        unsigned char b8:1;
        unsigned char b9:1;
        unsigned char b10:1;
        unsigned char b11:1;
        unsigned char b12:1;
        unsigned char b13:1;
        unsigned char b14:1;
        unsigned char b15:1;
        unsigned char b16:1;
        unsigned char b17:1;
        unsigned char b18:1;
        unsigned char b19:1;
        unsigned char b20:1;
        unsigned char b21:1;
        unsigned char b22:1;
        unsigned char b23:1;
        unsigned char b24:1;
        unsigned char b25:1;
        unsigned char b26:1;
        unsigned char b27:1;
        unsigned char b28:1;
        unsigned char b29:1;
        unsigned char b30:1;
        unsigned char b31:1;
    } bits;
} DWORD_VAL;
#endif //__GENERIC_TYPE_DEFS_H_
```

HardwareProfile.h

```
#ifndef _HARDWARE_PROFILE_H
#define _HARDWARE_PROFILE_H
#include "GenericTypeDefs.h"
BYTE ButtonPressed(void);
void BoardInit(void);
#endif
```

MSPI.h

```
#ifndef _SPI_H_
#define _SPI_H_
/********************** HEADERS *********************************/
#include "Compiler.h"
#include "GenericTypeDefs.h"
/********************** FUNCTION PROTOTYPES *********************/
void SPIPut(BYTE v);
BYTE SPIGet(void);
/********************** MACROS **********************************/
#define SPIInit()   SSPIF = 1
#endif
```

SystemProfile.h

```
#include "P2PDefs.h"
```

SimpleExampleNode2.h (Contador de Tempo de Bilhar)

```
#define EEP_TAR01H (0x0A)
#define EEP_TAR01L (0x0B)
// globals.h
#ifdef _GLOBALS_C
  #define EXTERN
  #define INIT(a)   = a
#else
  #define EXTERN extern
  #define INIT(a)
#endif
EXTERN   int tarifa;
```

MRF24J40.h

```
#if !defined(_ZMRF24J40_H_)
#define _ZMRF24J40_H_
//long address registers
#define RFCTRL0 (0x200)
#define RFCTRL1 (0x201)
#define RFCTRL2 (0x202)
#define RFCTRL3 (0x203)
#define RFCTRL4 (0x204)
#define RFCTRL5 (0x205)
#define RFCTRL6 (0x206)
#define RFCTRL7 (0x207)
#define RFCTRL8 (0x208)
#define CAL1 (0x209)
#define CAL2 (0x20a)
#define CAL3 (0x20b)
#define SFCNTRH (0x20c)
#define SFCNTRM (0x20d)
#define SFCNTRL (0x20e)
#define RFSTATE (0x20f)
#define RSSI (0x210)
#define CLKIRQCR (0x211)
#define SRCADRMODE (0x212)
#define SRCADDR0 (0x213)
#define SRCADDR1 (0x214)
#define SRCADDR2 (0x215)
#define SRCADDR3 (0x216)
#define SRCADDR4 (0x217)
#define SRCADDR5 (0x218)
#define SRCADDR6 (0x219)
#define SRCADDR7 (0x21a)
#define RXFRAMESTATE (0x21b)
#define SECSTATUS (0x21c)
#define STCCMP (0x21d)
#define HLEN (0x21e)
#define FLEN (0x21f)
#define SCLKDIV (0x220)
//#define reserved (0x221)
#define WAKETIMEL (0x222)
#define WAKETIMEH (0x223)
#define TXREMCNTL (0x224)
#define TXREMCNTH (0x225)
#define TXMAINCNTL (0x226)
#define TXMAINCNTM (0x227)
#define TXMAINCNTH0 (0x228)
#define TXMAINCNTH1 (0x229)
#define RFMANUALCTRLEN (0x22a)
#define RFMANUALCTRL (0x22b)
#define RFRXCTRL RFMANUALCTRL
#define TxDACMANUALCTRL (0x22c)
#define RFMANUALCTRL2 (0x22d)
#define TESTRSSI (0x22e)
#define TESTMODE (0x22f)
#define NORMAL_TX_FIFO  (0x000)
#define BEACON_TX_FIFO  (0x080)
#define GTS1_TX_FIFO    (0x100)
#define GTS2_TX_FIFO    (0x180)
#define RX_FIFO         (0x300)
#define SECURITY_FIFO   (0x280)
//short address registers for reading
#define READ_RXMCR (0x00)
#define READ_PANIDL (0x02)
#define READ_PANIDH (0x04)
#define READ_SADRL (0x06)
#define READ_SADRH (0x08)
#define READ_EADR0 (0x0A)
#define READ_EADR1 (0x0C)
#define READ_EADR2 (0x0E)
```

```
#define READ_EADR3 (0x10)
#define READ_EADR4 (0x12)
#define READ_EADR5 (0x14)
#define READ_EADR6 (0x16)
#define READ_EADR7 (0x18)
#define READ_RXFLUSH (0x1a)
#define READ_TXSTATE0 (0x1c)
#define READ_TXSTATE1 (0x1e)
#define READ_ORDER (0x20)
#define READ_TXMCR (0x22)
#define READ_ACKTMOUT (0x24)
#define READ_SLALLOC (0x26)
#define READ_SYMTICKL (0x28)
#define READ_SYMTICKH (0x2A)
#define READ_PAONTIME (0x2C)
#define READ_PAONSETUP (0x2E)
#define READ_FFOEN (0x30)
#define READ_CSMACR (0x32)
#define READ_TXBCNTRIG (0x34)
#define READ_TXNMTRIG (0x36)
#define READ_TXG1TRIG (0x38)
#define READ_TXG2TRIG (0x3A)
#define READ_ESLOTG23 (0x3C)
#define READ_ESLOTG45 (0x3E)
#define READ_ESLOTG67 (0x40)
#define READ_TXPEND (0x42)
#define READ_TXBCNINTL (0x44)
#define READ_FRMOFFSET (0x46)
#define READ_TXSR (0x48)
#define READ_TXLERR (0x4A)
#define READ_GATE_CLK (0x4C)
#define READ_TXOFFSET (0x4E)
#define READ_HSYMTMR0 (0x50)
#define READ_HSYMTMR1 (0x52)
#define READ_SOFTRST (0x54)
#define READ_BISTCR (0x56)
#define READ_SECCR0 (0x58)
#define READ_SECCR1 (0x5A)
#define READ_TXPEMISP (0x5C)
#define READ_SECISR (0x5E)
#define READ_RXSR (0x60)
#define READ_ISRSTS (0x62)
#define READ_INTMSK (0x64)
#define READ_GPIO (0x66)
#define READ_GPIODIR (0x68)
#define READ_SLPACK (0x6A)
#define READ_RFCTL (0x6C)
#define READ_SECCR2 (0x6E)
#define READ_BBREG0 (0x70)
#define READ_BBREG1 (0x72)
#define READ_BBREG2 (0x74)
#define READ_BBREG3 (0x76)
#define READ_BBREG4 (0x78)
#define READ_BBREG5 (0x7A)
#define READ_BBREG6 (0x7C)
#define READ_RSSITHCCA (0x7E)
//short address registers for writing
//short address registers for reading
#define WRITE_RXMCR (0x01)
#define WRITE_PANIDL (0x03)
#define WRITE_PANIDH (0x05)
#define WRITE_SADRL (0x07)
#define WRITE_SADRH (0x09)
#define WRITE_EADR0 (0x0B)
#define WRITE_EADR1 (0x0D)
#define WRITE_EADR2 (0x0F)
#define WRITE_EADR3 (0x11)
#define WRITE_EADR4 (0x13)
#define WRITE_EADR5 (0x15)
#define WRITE_EADR6 (0x17)
#define WRITE_EADR7 (0x19)
#define WRITE_RXFLUSH (0x1B)
#define WRITE_TXSTATE0 (0x1D)
#define WRITE_TXSTATE1 (0x1F)
#define WRITE_ORDER (0x21)
```

```
#define WRITE_TXMCR (0x23)
#define WRITE_ACKTMOUT (0x25)
#define WRITE_SLALLOC (0x27)
#define WRITE_SYMTICKL (0x29)
#define WRITE_SYMTICKH (0x2B)
#define WRITE_PAONTIME (0x2D)
#define WRITE_PAONSETUP (0x2F)
#define WRITE_FFOEN (0x31)
#define WRITE_CSMACR (0x33)
#define WRITE_TXBCNTRIG (0x35)
#define WRITE_TXNMTRIG (0x37)
#define WRITE_TXG1TRIG (0x39)
#define WRITE_TXG2TRIG (0x3B)
#define WRITE_ESLOTG23 (0x3D)
#define WRITE_ESLOTG45 (0x3F)
#define WRITE_ESLOTG67 (0x41)
#define WRITE_TXPEND (0x43)
#define WRITE_TXBCNINTL (0x45)
#define WRITE_FRMOFFSET (0x47)
#define WRITE_TXSR (0x49)
#define WRITE_TXLERR (0x4B)
#define WRITE_GATE_CLK (0x4D)
#define WRITE_TXOFFSET (0x4F)
#define WRITE_HSYMTMR0 (0x51)
#define WRITE_HSYMTMR1 (0x53)
#define WRITE_SOFTRST (0x55)
#define WRITE_BISTCR (0x57)
#define WRITE_SECCR0 (0x59)
#define WRITE_SECCR1 (0x5B)
#define WRITE_TXPEMISP (0x5D)
#define WRITE_SECISR (0x5F)
#define WRITE_RXSR (0x61)
#define WRITE_ISRSTS (0x63)
#define WRITE_INTMSK (0x65)
#define WRITE_GPIO (0x67)
#define WRITE_GPIODIR (0x69)
#define WRITE_SLPACK (0x6B)
#define WRITE_RFCTL (0x6D)
#define WRITE_SECCR2 (0x6F)
#define WRITE_BBREG0 (0x71)
#define WRITE_BBREG1 (0x73)
#define WRITE_BBREG2 (0x75)
#define WRITE_BBREG3 (0x77)
#define WRITE_BBREG4 (0x79)
#define WRITE_BBREG5 (0x7B)
#define WRITE_BBREG6 (0x7D)
#define WRITE_RSSITHCCA (0x7F)
#define CHANNEL_11 0x00
#define CHANNEL_12 0x10
#define CHANNEL_13 0x20
#define CHANNEL_14 0x30
#define CHANNEL_15 0x40
#define CHANNEL_16 0x50
#define CHANNEL_17 0x60
#define CHANNEL_18 0x70
#define CHANNEL_19 0x80
#define CHANNEL_20 0x90
#define CHANNEL_21 0xa0
#define CHANNEL_22 0xb0
#define CHANNEL_23 0xc0
#define CHANNEL_24 0xd0
#define CHANNEL_25 0xe0
#define CHANNEL_26 0xf0
#endif
```

P2P.h

```
#ifndef __P2P_H_
#define __P2P_H_
#define INPUT
#define OUTPUT
#define IOPUT
/************************ HEADERS **********************************/
#include "P2PDefs.h"
#include "SymbolTime.h"
/************************ DEFINITIONS ****************************/
#define STATUS_SUCCESS              0x00
#define STATUS_EXISTS               0x01
#define STATUS_ACTIVE_SCAN          0x02
#define STATUS_ENTRY_NOT_EXIST      0xF0
#define STATUS_NOT_ENOUGH_SPACE     0xF1
#define STATUS_NOT_SAME_PAN         0xF2
#define CMD_P2P_CONNECTION_REQUEST              0x81
#define CMD_P2P_CONNECTION_REMOVAL_REQUEST      0x82
#define CMD_DATA_REQUEST                        0x83
#define CMD_CHANNEL_HOPPING                     0x84
#define CMD_TIME_SYNCHRONIZATION_REQUEST        0x85
#define CMD_TIME_SYNCHRONIZATION_NOTIFICATION   0x86
#define CMD_P2P_CONNECTION_RESPONSE             0x91
#define CMD_P2P_CONNECTION_REMOVAL_RESPONSE     0x92
#define CMD_MAC_DATA_REQUEST                    0x04
#define RFD_DATA_WAIT 0x00003FFF
#ifdef ENABLE_FREQUENCY_AGILITY
   #define FA_BROADCAST_TIME          0x03
   #define CCA_FAILURE_TIMES          0x03
   #define ACK_FAILURE_TIMES          0x03
   #define RESYNC_TIMES               0x03
#endif

#if defined(ENABLE_FREQUENCY_AGILITY) || defined(ENABLE_SLEEP)
   #define VERIFY_TRANSMIT
#endif

/************************ DATA TYPE ******************************/
 /*********************************************************************
 * Overview: The interpolation of MRF24J40 radio interrupts. Details of
 * interrupts can be found in MRF24J40 data sheet
 *********************************************************************/
typedef union
{
   BYTE Val;               // value of interrupts
   struct
   {
      BYTE RF_TXIF :1;     // transmission finish interrupt
      BYTE :2;
      BYTE RF_RXIF :1;     // receiving a packet interrupt
      BYTE SECIF :1;       // receiving a secured packet interrupt
      BYTE :4;
   }bits;                  // bit map of interrupts
} MRF24J40_IFREG;
/*********************************************************************
 * Overview: The record to store the information of peer device in the
 * MiWi(TM) P2P Connection
 *********************************************************************/
typedef struct
{
   BYTE       PeerLongAddress[8];         // the long address of the peer device
   #if defined(ENABLE_SECURITY) && !defined(TARGET_SMALL)
      DWORD_VAL   IncomingFrameCounter;    // the incoming frame counter. Used to check
                                           // frame freshness to avoid repeat attack
                                           // only valid if security is enabled
   #endif
   BYTE       PeerInfo[1+ADDITIONAL_CONNECTION_PAYLOAD];  // The peer information. The first byte
                                                          // is the standard peer information. The
```

```
                                        // detailed definition of the first byte
                                        // can be found in P2P_CAPACITY definition.
                                        // Additional info can be added according
                                        // to the specific application
} P2P_CONNECTION_ENTRY;

/*****************************************************************
 * Overview: The capacity information for a MiWi(TM) P2P device. It is the
 * definition of the first byte of PeerInfo defined in
 * P2P_CONNECTION_ENTRY. The highest bit also be used to indicate
 * if the P2P connection entry is a valid entry
 *****************************************************************/
typedef union
{
    BYTE    Val;                    // the value of the P2P capacitity
    struct _P2P_CAPACITY_BITS
    {
        BYTE    RXOnWhileIdel      : 1;    // if device turns on radio when idle
        BYTE    DataRequestNeeded  : 1;    // if data request is required when device turns off radio when
                                           // idle. It is used to decide if an indirect message is necessary
                                           // to be stored.
        BYTE    TimeSynchronization : 1;    // reserved bit for future development
        BYTE    SecurityCapacity   : 1;    // if the device is capable of handling encrypted information
        BYTE    filler             : 3;
        BYTE    isValid            : 1;    // use this bit to indicate that this entry is a valid entry
    } bits;
} P2P_CAPACITY;
/*****************************************************************
 * Overview: The structure that stores all information about the received
 * frame, provided to the upper layer of the MiWi(TM) P2P stack
 *****************************************************************/
typedef struct
{
    union
    {
        BYTE    Val;                    // value of the flags for RECEIVED_FRAME
        struct
        {
            BYTE    commandFrame    : 1;    // if the received frame a command frame. data: 0; command 1
            BYTE    security        : 1;    // if the received frame is encrypted during transmission.
                                            // The payload in this structure has already been decrypted
            BYTE    framePending    : 1;    // if the frame pending bit has been set in the frame control
                                            // this bit is hardly used, just a legancy inherited from
                                            // IEEE 802.15.4
            BYTE    intraPAN        : 1;    // if the received frame a intra-PAN frame. Meaning if the
                                            // the source PAN ID the same as the destination PAN ID
            BYTE    broadcast       : 1;    // if the received frame a broadcast message
        } bits;                             // bit map of the flags for received frame
    } flags;                                // received frame flags

    #ifndef TARGET_SMALL
        BYTE        PacketLQI;              // the link quality indication for the received packet
                                            // that indicats the quality of the received frame
        BYTE        PacketRSSI;             // the RSSI of for the received packet that indicates
                                            // the signal strength of the received frame
        WORD_VAL    SourcePANID;            // The PAN identifier of the source device
    #endif
    BYTE        SourceLongAddress[8];       // The long address of the source device
    BYTE        PayLoadSize;                // The size of the payload
    BYTE        *PayLoad;                   // The pointer to the pay load of the received packet.
                                            // The pay load is the MAC payload without the MAC header
} RECEIVED_FRAME;
/*****************************************************************
 * Overview: The structure to store indirect messages for devices turn off
 * radio when idle
 *****************************************************************/
typedef struct
{
    TICK        TickStart;      // start time of the indirect message. Used for checking
                                // indirect message time out
    WORD_VAL    DestPANID;      // the PAN identifier for the destination node
    union
    {
        BYTE        DestLongAddress[8];             // unicast destination long address
        BYTE        DestIndex[P2P_CONNECTION_SIZE]; // broadcast index of the P2P Connection Entries
```

```
                                    // for destination RFD devices
    } DestAddress;  // destination address for the indirect message. Can either for unicast or broadcast
    union
    {
        BYTE    Val;                    // value for the flags
        struct
        {
            BYTE    isValid         : 1;    // if this indirect message is valid
            BYTE    isBroadcast     : 1;    // if this indirect message is for broadcasting
            BYTE    isCommand       : 1;    // if this indirect message a command
            BYTE    isSecured       : 1;    // if this indirect message requires encryption
        } bits;                         // bit map of the flags
    } flags;                            // flags for indirect message
    BYTE        PayLoadSize;            // the indirect message pay load size
    BYTE        PayLoad[TX_BUFFER_SIZE-21]; // the indirect message pay load
} INDIRECT_MESSAGE;
/************************************************************************
 * Overview: structure to indicate the status of P2P stack
 ************************************************************************/
typedef union
{
    WORD Val;                           // The value of the P2P status flags
    struct
    {
        BYTE RX_PENDING             :1;     // indicate if a frame is received and pending to read
        BYTE RX_BUFFERED            :1;     // indicate if a frame has been read into the buffer
                                    // of MCU and waiting to be processed
        BYTE RX_ENABLED             :1;     // indicate if the radio is enabled to receive the
                                    // next packet
        BYTE RX_IGNORE_SECURITY     :1;     // indicate if we need to ignore the current packet
                                    // with encryption. Usually, the reason is frame counter
                                    // freshness checking fails
        BYTE RX_SECURITY            :1;     // indicate if current received packet is encrypted
        BYTE TX_BUSY                :1;     // indicate if the radio is currently busy transmitting
                                    // a packet
        BYTE TX_PENDING_ACK         :1;     // indicate if the current transmitting packet waiting
                                    // for an acknowledgement from the peer device
        BYTE PHY_SLEEPING           :1;     // indicate if the device in sleeping state
        BYTE TimeToSleep            :1;     // used by the application layer to indicate that the RFD
                                    // device is idle and ready to go to sleep
        BYTE DataRequesting         :1;     // indicate that device is in the process of data request
                                    // from its parent. Only effective if device enables sleeping
        BYTE RxHasUserData          :1;     // indicate if the received frame needs processing from
                                    // the application layer
        BYTE AckRequired            :1;     // indicate if the transmitting packet require acknowledgement
                                    // from the peer device
        BYTE EnableNewConnection    :1;     // indicate if the current device allow new P2P connection
                                    // established.
        BYTE TX_FAIL                :1;     // indicate if the current transmission fails

        BYTE SearchConnection       :1;     // indicate if the stack is currently in the process of
                                    // looking for new connection
        BYTE Resync                 :1;     // indicate if the stack is currently in the process of
                                    // resynchronizing connection with the peer device
    }bits;                              // bit map of the P2P status
} P2P_STATUS;

#ifdef ENABLE_ACTIVE_SCAN
    /************************************************************************
     * Overview: The structure that store the active scan result
     ************************************************************************/
    typedef struct
    {
        BYTE        Channel;        // The channel of the PAN that is operating on
        BYTE        RSSIValue;      // The signal strength of the PAN. If there are more than one
                                    // devices that belongs to the same PAN and respond to the active
                                    // scan, the one with the strongest signal will be recorded
        WORD_VAL    PANID;          // The PAN identifier of the PAN
    } ACTIVE_SCAN_RESULT;
#endif

/********************** EXTERNAL VARIABLES **********************/
extern BYTE TxBuffer[];
extern BYTE TxData;
extern WORD_VAL myPANID;
```

```
extern P2P_CONNECTION_ENTRY    P2PConnections[P2P_CONNECTION_SIZE];
extern volatile P2P_STATUS P2PStatus;
extern RECEIVED_FRAME rxFrame;

#ifdef ENABLE_ACTIVE_SCAN
    #define ACTIVE_SCAN_RESULT_SIZE 16
    extern ACTIVE_SCAN_RESULT ActiveScanResults[ACTIVE_SCAN_RESULT_SIZE];
#endif

#ifdef ENABLE_FREQUENCY_AGILITY
    extern BYTE AckFailureTimes;
    extern BYTE CCAFailureTimes;
    extern BYTE currentChannel;
#endif
/************************ MACROS ********************************/

/*********************************************************************
 * Write one byte into transmission buffer
 ********************************************************************/
#define WriteData(a) TxBuffer[TxData++] = a

/*********************************************************************
 * Reset the transmission buffer before writing any data
 ********************************************************************/
#define FlushTx() TxData = 0;

/*********************************************************************
 * Enable the MiWi(TM) P2P stack to accept new P2P connection
 * request
 ********************************************************************/
#define EnableNewConnection() P2PStatus.bits.EnableNewConnection = 1;

/*********************************************************************
 * Disable the MiWi(TM) P2P stack to accept new P2P connection
 * request
 ********************************************************************/
#define DisableNewConnection() P2PStatus.bits.EnableNewConnection = 0;

/*********************************************************************
 * Enable the request of acknowledgment for all unicast message
 ********************************************************************/
#define EnableAcknowledgement() P2PStatus.bits.AckRequired = 1;

/*********************************************************************
 * Disable the request of acknowledgement for all unicast message
 ********************************************************************/
#define DisableAcknowledgement() P2PStatus.bits.AckRequired = 0;

/*********************************************************************
 * Notify the P2P stack that a data request procedure is going on
 ********************************************************************/
#define DataRequesting()    P2PStatus.bits.DataRequesting = 1;

/*********************************************************************
 * Check if the stack is in the data request process
 ********************************************************************/
#define isDataRequesting()  P2PStatus.bits.DataRequesting

/*********************************************************************
 * Notify the stack that the device is idle and ready to go to
 * sleep now
 ********************************************************************/
#define ReadyToSleep()      P2PStatus.bits.TimeToSleep = 1;

/*********************************************************************
 * Check if the stack is idle and ready to go to sleep
 ********************************************************************/
#define isReadyToSleep()    P2PStatus.bits.TimeToSleep

/*********************************************************************
 * Clear the flag to notify the stack that the device is idle
 * and ready to go to sleep
 ********************************************************************/
#define ClearReadyToSleep() P2PStatus.bits.TimeToSleep = 0;
```

```
/********************** FUNCTION PROTOTYPES *********************/
void P2PInit(void);
void P2PTasks(void);
void DiscardPacket(void);
void SetChannel(INPUT BYTE channel);
void initMRF24J40(void);

void PHYSetLongRAMAddr(INPUT WORD address, INPUT BYTE value);
void PHYSetShortRAMAddr(INPUT BYTE address, INPUT BYTE value);
BYTE PHYGetShortRAMAddr(INPUT BYTE address);
BYTE PHYGetLongRAMAddr(INPUT WORD address);

BOOL BroadcastPacket( INPUT WORD_VAL DestinationPANID, INPUT BOOL isCommand, INPUT BOOL SecurityEnabled );
BOOL UnicastConnection( INPUT BYTE ConnectionIndex, INPUT BOOL isCommand, INPUT BOOL SecurityEnabled);
BOOL UnicastLongAddress( INPUT WORD_VAL DestinationPANID, INPUT BYTE *DestinationAddress, INPUT BOOL
isCommand, INPUT BOOL SecurityEnabled);
void DumpConnection(INPUT BYTE index);
BOOL ReceivedPacket(void);

#ifdef ENABLE_ACTIVE_SCAN
   BYTE ActiveScan(INPUT BYTE ScanDuration, INPUT DWORD ChannelMap);
#endif

#ifdef ENABLE_ED_SCAN
   BYTE CreateNewConnection(INPUT BYTE RetryInterval, INPUT DWORD ChannelMap);
   BYTE OptimalChannel(INPUT BYTE ScanDuration, INPUT DWORD ChannelMap, OUTPUT BYTE *RSSIValue);
#else
   BYTE CreateNewConnection(INPUT BYTE RetryInterval);
#endif

#ifdef ENABLE_FREQUENCY_AGILITY
   BOOL InitChannelHopping( INPUT DWORD ChannelMap);
   BOOL ResyncConnection(INPUT BYTE *DestinationAddress, INPUT DWORD ChannelMap);
#endif

#ifdef ENABLE_SLEEP
   void MRF24J40Sleep(void);
   void MRF24J40Wake(void);
   BOOL CheckForData(void);
#endif
#endif
```

P2PDefs.h

```
#ifndef _P2P_DEFS_H_
#define _P2P_DEFS_H_

/************************************************************************/
// ENABLE_CONSOLE will enable the print out on the hyper terminal
// this definition is very helpful in the debugging process
/************************************************************************/
#define ENABLE_CONSOLE

/************************************************************************/
// following codes defines the platforms as well as the hardware
// configuration
/************************************************************************/
#if defined(__18CXX)
   #define PICDEMZ
#endif

#if defined(__dsPIC30F__) || defined(__dsPIC33F__) || defined(__PIC24F__) || defined(__PIC24H__) ||
defined(__PIC32MX__)
   #define EXPLORER16
#endif

#if defined(PICDEMZ)
   #if defined(__dsPIC30F__) || defined(__dsPIC33F__) || defined(__dsPIC24F__) || defined(__dsPIC24H__) ||
defined(__PIC32MX__)
      #error "Incorrect board/processor combination."
   #endif
#endif

#if defined(EXPLORER16)
   #if defined(__18CXX)
      #error "Incorrect board/processor combination."
   #endif
#endif


   #define CLOCK_FREQ 16000000
   // Transceiver Configuration
   #define RFIF INTCONbits.INT0IF
   #define RFIE INTCONbits.INT0IE
   #define PHY_CS LATCbits.LATC0
   #define PHY_CS_TRIS TRISCbits.TRISC0
   #define PHY_RESETn LATCbits.LATC2
   #define PHY_RESETn_TRIS TRISCbits.TRISC2
   #define PHY_WAKE LATCbits.LATC1
   #define PHY_WAKE_TRIS TRISCbits.TRISC1

   #define PUSH_BUTTON_1 PORTBbits.RB5
   #define PUSH_BUTTON_2 PORTBbits.RB4
   #define LED_1 PORTAbits.RA0
   #define LED_2 PORTAbits.RA1

   #define PUSH_BUTTON_1_TRIS TRISB5
   #define PUSH_BUTTON_2_TRIS TRISB4
   #define LED_1_TRIS TRISA0
   #define LED_2_TRIS TRISA1

   #define RF_INT_PIN PORTBbits.RB0
   #define TMRL TMR0L

// MAC Address
#define EUI_7 0x11
#define EUI_6 0x22
#define EUI_5 0x33
#define EUI_4 0x44
#define EUI_3 0x55
#define EUI_2 0x66
```

```
#define EUI_1 0x77
#define EUI_0 0x03

// PICmicro Information
#define BAUD_RATE 19200
#define FREQUENCY_BAND 2400

// Message Buffers
#define TX_BUFFER_SIZE 127
#define RX_BUFFER_SIZE 127

#define SECURITY_KEY_00 0x00
#define SECURITY_KEY_01 0x01
#define SECURITY_KEY_02 0x02
#define SECURITY_KEY_03 0x03
#define SECURITY_KEY_04 0x04
#define SECURITY_KEY_05 0x05
#define SECURITY_KEY_06 0x06
#define SECURITY_KEY_07 0x07
#define SECURITY_KEY_08 0x08
#define SECURITY_KEY_09 0x09
#define SECURITY_KEY_10 0x0a
#define SECURITY_KEY_11 0x0b
#define SECURITY_KEY_12 0x0c
#define SECURITY_KEY_13 0x0d
#define SECURITY_KEY_14 0x0e
#define SECURITY_KEY_15 0x0f
#define KEY_SEQUENCE_NUMBER 0x00
#define SECURITY_LEVEL 0x04

/*****************************************************************************/
// MY_PAN_ID defines the PAN identifier
/*****************************************************************************/
#define MY_PAN_ID                   0x1234

/*****************************************************************************/
// ADDITIONAL_CONNECTION_PAYLOAD defines the size of additional payload
// will be attached to the P2P Connection Request. Additional payload
// is the information that the devices what to share with their peers
// on the P2P connection. The additional payload will be defined by
// the application and defined in main.c
/*****************************************************************************/
#define ADDITIONAL_CONNECTION_PAYLOAD   0

/*****************************************************************************/
// P2P_CONNECTION_SIZE defines the maximum P2P connections that this
// device allowes at the same time.
/*****************************************************************************/
#define P2P_CONNECTION_SIZE           10

/*****************************************************************************/
// P2P_CAPACITY_INFO defines the capability of current device. The
// capability is defined as the bitmap of following definition:
// ---------------------------------------------------------------------------
// |    Bit 0    |      Bit 1      |     Bit 2     |     Bit 3      | Bit 4 - 7|
// ---------------------------------------------------------------------------
// | RXOnWhenIdle | DataRequestNeeded | TimeSynRequired | SecurityEnabled | Reserved |
// ---------------------------------------------------------------------------
/*****************************************************************************/
#define P2P_CAPACITY_INFO             0x01

/*****************************************************************************/
// TARGET_SMALL will remove the support of inter PAN communication
// and other minor features to save programming space
/*****************************************************************************/
//#define TARGET_SMALL

/*****************************************************************************/
// ENABLE_DUMP will enable the stack to be able to print out the
// content of the P2P connection entry. It is useful in the debugging
// process
/*****************************************************************************/
#define ENABLE_DUMP

/*****************************************************************************/
```

```
// ENABLE_SLEEP will enable the device to go to sleep and wake up
// from the sleep
/*****************************************************************/
#ifndef SIMPLE_EXAMPLE
   //#define ENABLE_SLEEP
#endif

/*****************************************************************/
// ENABLE_ED_SCAN will enable the device to do an energy detection scan
// to find out the channel with least noise and operate on that channel
/*****************************************************************/
#ifndef SIMPLE_EXAMPLE
   #define ENABLE_ED_SCAN
#endif

/*****************************************************************/
// ENABLE_ACTIVE_SCAN will enable the device to do an active scan to
// to detect current existing connection.
/*****************************************************************/
#ifndef SIMPLE_EXAMPLE
   #define ENABLE_ACTIVE_SCAN
#endif

/*****************************************************************/
// ENABLE_SECURITY will enable the device to encrypt and decrypt
// information transferred
/*****************************************************************/
#define ENABLE_SECURITY

/*****************************************************************/
// ENABLE_INDIRECT_MESSAGE will enable the device to store the packets
// for the sleeping devices temporily until they wake up and ask for
// the messages
/*****************************************************************/
#ifndef SIMPLE_EXAMPLE
   #define ENABLE_INDIRECT_MESSAGE
#endif

/*****************************************************************/
// ENABLE_BROADCAST will enable the device to broadcast messages for
// the sleeping devices until they wake up and ask for the messages
/*****************************************************************/
#ifndef SIMPLE_EXAMPLE
   #define ENABLE_BROADCAST
#endif

/*****************************************************************/
// INDIRECT_MESSAGE_SIZE defines the maximum number of packets that
// the device can store for the sleeping device(s)
/*****************************************************************/
#ifndef SIMPLE_EXAMPLE
   #define INDIRECT_MESSAGE_SIZE   2
#endif

/*****************************************************************/
// INDIRECT_MESSAGE_TIMEOUT defines the timeout interval in seconds
// for the stored packets for sleeping devices
/*****************************************************************/
#ifndef SIMPLE_EXAMPLE
   #define INDIRECT_MESSAGE_TIMEOUT 10
#endif

/*****************************************************************/
// ENABLE_FREQUENCY_AGILITY will enable the device to change operating
// channel to bypass the sudden change of noise
/*****************************************************************/
#ifndef SIMPLE_EXAMPLE
   #define ENABLE_FREQUENCY_AGILITY
#endif

/*****************************************************************/
// FREQUENCY_AGILITY_STARTER defines the device with the capability
// of starting the process of channel hopping. The device that can
// start the channel hopping must be able to do an energy scan
// to decide the optimal channel
```

```
/*****************************************************************/
#ifndef SIMPLE_EXAMPLE
   #define FREQUENCY_AGILITY_STARTER
#endif

#if defined(ENABLE_ACTIVE_SCAN) && defined(TARGET_SMALL)
   #error  Target_Small and Enable_Active_Scan cannot be defined together
#endif

#if defined(FREQUENCY_AGILITY_STARTER) && !defined(ENABLE_ED_SCAN)
   #error The FREQUENCY_AGILITY_STARTER cannot be defined if energy scan is not enabled
#endif

// Constants Validation

// ZENA(TM) will automatically range check the entered values.  These range
// checks are included here in cases the application developer manually
// adjusts the values.

#if (RX_BUFFER_SIZE > 127)
   #error RX BUFFER SIZE too large. Must be <= 127.
#endif

#if (TX_BUFFER_SIZE > 127)
   #error TX BUFFER SIZE too large. Must be <= 127.
#endif

#if (RX_BUFFER_SIZE < 25)
   #error RX BUFFER SIZE too small. Must be >= 25.
#endif

#if (TX_BUFFER_SIZE < 25)
   #error TX BUFFER SIZE too small. Must be >= 25.
#endif

#if (NETWORK_TABLE_SIZE > 0xFE)
   #error NETWORK TABLE SIZE too large.  Must be < 0xFF.
#endif

#if (INDIRECT_BUFFER_SIZE > 0xFE)
   #error INDIRECT BUFFER SIZE too large.  Must be < 0xFF.
#endif
#endif
```

SymbolTime.h

```
#ifndef __SYMBOL_TIME_H_
#define __SYMBOL_TIME_H_

/************************ HEADERS ***********************************/
#include "P2PDefs.h"
#include "Compiler.h"
#include "GenericTypeDefs.h"
/************************ DEFINITIONS ******************************/
#if defined(__18CXX)
    /* this section is based on the Timer 0 module of the PIC18 family */
    #if(FREQUENCY_BAND == 2400)
        #if(CLOCK_FREQ <= 250000)
            #define CLOCK_DIVIDER 1
            #define CLOCK_DIVIDER_SETTING 0x08 /* no prescalar */
            #define SYMBOL_TO_TICK_RATE 250000
        #elif(CLOCK_FREQ <= 500000)
            #define CLOCK_DIVIDER 2
            #define CLOCK_DIVIDER_SETTING 0x00
            #define SYMBOL_TO_TICK_RATE 500000
        #elif(CLOCK_FREQ <= 1000000)
            #define CLOCK_DIVIDER 4
            #define CLOCK_DIVIDER_SETTING 0x01
            #define SYMBOL_TO_TICK_RATE 1000000
        #elif(CLOCK_FREQ <= 2000000)
            #define CLOCK_DIVIDER 8
            #define CLOCK_DIVIDER_SETTING 0x02
            #define SYMBOL_TO_TICK_RATE 2000000
        #elif(CLOCK_FREQ <= 4000000)
            #define CLOCK_DIVIDER 16
            #define CLOCK_DIVIDER_SETTING 0x03
            #define SYMBOL_TO_TICK_RATE 4000000
        #elif(CLOCK_FREQ <= 8000000)
            #define CLOCK_DIVIDER 32
            #define CLOCK_DIVIDER_SETTING 0x04
            #define SYMBOL_TO_TICK_RATE 8000000
        #elif(CLOCK_FREQ <= 16000000)
            #define CLOCK_DIVIDER 64
            #define CLOCK_DIVIDER_SETTING 0x05
            #define SYMBOL_TO_TICK_RATE 16000000
        #elif(CLOCK_FREQ <= 3200000)
            #define CLOCK_DIVIDER 128
            #define CLOCK_DIVIDER_SETTING 0x06
            #define SYMBOL_TO_TICK_RATE 32000000
        #else
            #define CLOCK_DIVIDER 256
            #define CLOCK_DIVIDER_SETTING 0x07
            #define SYMBOL_TO_TICK_RATE 32000000
        #endif
            #define ONE_SECOND ((CLOCK_FREQ/1000 * 62500) / (SYMBOL_TO_TICK_RATE / 1000))
        /* SYMBOLS_TO_TICKS to only be used with input (a) as a constant, otherwise you will blow up the code */
        #define SYMBOLS_TO_TICKS(a) ((CLOCK_FREQ/1000) / (SYMBOL_TO_TICK_RATE / 1000)) * a
    #endif
#elif defined(__dsPIC30F__) || defined(__dsPIC33F__) || defined(__PIC24F__) || defined(__PIC24H__)
    /* this section is based on the Timer 2/3 module of the dsPIC33/PIC24 family */
    #if(FREQUENCY_BAND == 2400)
        #if(CLOCK_FREQ <= 125000)
            #define CLOCK_DIVIDER 1
            #define CLOCK_DIVIDER_SETTING 0x0000 /* no prescalar */
            #define SYMBOL_TO_TICK_RATE 125000
        #elif(CLOCK_FREQ <= 1000000)
            #define CLOCK_DIVIDER 8
            #define CLOCK_DIVIDER_SETTING 0x0010
            #define SYMBOL_TO_TICK_RATE 1000000
        #elif(CLOCK_FREQ <= 8000000)
            #define CLOCK_DIVIDER 64
            #define CLOCK_DIVIDER_SETTING 0x0020
            #define SYMBOL_TO_TICK_RATE 8000000
```

```
        #else
            #define CLOCK_DIVIDER 256
            #define CLOCK_DIVIDER_SETTING 0x0030
            #define SYMBOL_TO_TICK_RATE 32000000
        #endif
            #define ONE_SECOND ((CLOCK_FREQ/1000 * 62500) / (SYMBOL_TO_TICK_RATE / 1000))
        /* SYMBOLS_TO_TICKS to only be used with input (a) as a constant, otherwise you will blow up the code */
        #define SYMBOLS_TO_TICKS(a) ((CLOCK_FREQ/1000 ) / (SYMBOL_TO_TICK_RATE / 1000)) * a
    #endif
#elif defined(__PIC32MX__)
    /* this section is based on the Timer 2/3 module of the PIC32MX family */
    #if(FREQUENCY_BAND == 2400)
        #if(INSTR_FREQ <= 125000)
            #define CLOCK_DIVIDER 1
            #define CLOCK_DIVIDER_SETTING 0x0000 /* no prescalar */
            #define SYMBOL_TO_TICK_RATE 125000
        #elif(INSTR_FREQ <= 1000000)
            #define CLOCK_DIVIDER 8
            #define CLOCK_DIVIDER_SETTING 0x0030
            #define SYMBOL_TO_TICK_RATE 1000000
        #elif(INSTR_FREQ <= 8000000)
            #define CLOCK_DIVIDER 64
            #define CLOCK_DIVIDER_SETTING 0x0060
            #define SYMBOL_TO_TICK_RATE 8000000
        #elif(INSTR_FREQ <= 16000000)
            #define CLOCK_DIVIDER 256
            #define CLOCK_DIVIDER_SETTING 0x0070
            #define SYMBOL_TO_TICK_RATE INSTR_FREQ
        #else
            #define CLOCK_DIVIDER 256
            #define CLOCK_DIVIDER_SETTING 0x70
            #define SYMBOL_TO_TICK_RATE INSTR_FREQ
        #endif
            #define ONE_SECOND (((DWORD)INSTR_FREQ/1000 * 62500) / (SYMBOL_TO_TICK_RATE / 1000))
        /* SYMBOLS_TO_TICKS to only be used with input (a) as a constant, otherwise you will blow up the code */
        #define SYMBOLS_TO_TICKS(a) (((DWORD)(INSTR_FREQ/1000) * a) / (SYMBOL_TO_TICK_RATE / 1000))
    #endif
#else
    #error "Unsupported processor.  New timing definitions required for proper operation"
#endif
#define TickGetDiff(a,b) (a.Val - b.Val)
/******************** DATA TYPES *******************************/
typedef union _TICK
{
    DWORD Val;
    struct _TICK_bytes
    {
        BYTE b0;
        BYTE b1;
        BYTE b2;
        BYTE b3;
    } byte;
    BYTE v[4];
    struct _TICK_words
    {
        WORD w0;
        WORD w1;
    } word;
} TICK;
void InitSymbolTimer(void);
TICK TickGet(void);
/******************** VARIABLES *******************************/
extern volatile BYTE timerExtension1,timerExtension2;
#endif
```

Delays.h

```
#ifndef __DELAYS_H
#define __DELAYS_H

/* PIC18 cycle-count delay routines.
 *
 *   Functions:
 *                        Delay1TCY()
 *             Delay10TCY()   // 17Cxx only
 *             Delay10TCYx()
 *             Delay100TCYx()
 *             Delay1KTCYx()
 *             Delay10KTCYx()
 */

/* For definition of Nop() */
#include <p18cxxx.h>

/* Delay of exactly 1 Tcy */
#define Delay1TCY() Nop()

#define PARAM_SCLASS auto

/* Delay of exactly 10 Tcy */
#define Delay10TCY() Delay10TCYx(1)

/* Delay10TCYx
 * Delay multiples of 10 Tcy
 * Passing 0 (zero) results in a delay of 2560 cycles.
 * The 18Cxxx version of this function supports the full range [0,255]
 * The 17Cxxx version supports [2,255] and 0.
 */
void Delay10TCYx(PARAM_SCLASS unsigned char);

/* Delay100TCYx
 * Delay multiples of 100 Tcy
 * Passing 0 (zero) results in a delay of 25,600 cycles.
 * The full range of [0,255] is supported.
 */
void Delay100TCYx(PARAM_SCLASS unsigned char);

/* Delay1KTCYx
 * Delay multiples of 1000 Tcy
 * Passing 0 (zero) results in a delay of 256,000 cycles.
 * The full range of [0,255] is supported.
 */
void Delay1KTCYx(PARAM_SCLASS unsigned char);

/* Delay10KTCYx
 * Delay multiples of 10,000 Tcy
 * Passing 0 (zero) results in a delay of 2,560,000 cycles.
 * The full range of [0,255] is supported.
 */
void Delay10KTCYx(PARAM_SCLASS unsigned char);

#endif
```

b) Aplicação em Visual Basic

```
Dim var_ajuda As Integer
Dim var_ler As String


'variaveis utilizadas na criação da trama a enviar
Dim var_ajuda_01 As String
Dim var_ajuda_02 As String
Dim var_ajuda_03 As String
Dim criacao_trama As String

'variaveis utilizadas para a rede
Dim trama_recebida As String
Dim trama_alterada As String
Dim trama_sn_ano As String
Dim trama_sn_mes As String
Dim trama_sn_num As String
Dim trama_sn_completo As String
Dim trama_val_a_pagar As String

'variaveis utilizadas para a recepção da leitura do contador
Dim trama_fraccao As String
Dim trama_power_down As String
Dim trama_tarifa01 As String
Dim trama_total_facturado As String
Dim trama_password As String

‘comando enviar trama manual
Private Sub Command1_Click()
Do While Len(Text3.Text) <> Int(txt_n_bits)
   Text3.Text = Text3.Text & "@"
Loop
MSComm1.Output = Text3.Text
End Sub

‘comando apagar histórico
Private Sub Command2_Click()
Text1.Text = ""
End Sub

‘comando definição modo de rede
Private Sub Command3_Click()
Text3.Text = "!R"
End Sub

‘comando ler -> leitura individual
Private Sub Command4_Click()
var_ler = "$%"
Do While Len(var_ler) <> Int(txt_n_bits)
   var_ler = var_ler & "@"
Loop
MSComm1.Output = var_ler
End Sub

‘comando definição modo normal
Private Sub Command5_Click()
Text3.Text = "!N"
End Sub

‘comando enviar -> leitura individual
Private Sub Command6_Click()
'verificar se estão todos os campos preenchidos
If ((txt_sn <> "") And (txt_fraccao <> "") And (txt_power_down <> "") And (txt_tarifa01 <> "") And (txt_total_facturado
<> "") And (txt_pass <> "")) Then

   criacao_trama = ""
   'adiciona o inicio de trama
   criacao_trama = "$#"
```

```
    'adiciona o serial number
    var_ajuda01 = Left(txt_sn.Text, 2)
    var_ajuda02 = Left(Right(txt_sn.Text, Len(txt_sn.Text) - 2), 2)
    var_ajuda03 = Right(txt_sn.Text, Len(txt_sn.Text) - 5)
    Do While (Len(var_ajuda01) < 3)
       var_ajuda01 = "0" & var_ajuda01
    Loop
    Do While (Len(var_ajuda02) < 3)
       var_ajuda02 = "0" & var_ajuda02
    Loop
    Do While (Len(var_ajuda03) < 3)
       var_ajuda03 = "0" & var_ajuda03
    Loop
    criacao_trama = criacao_trama & var_ajuda01 & var_ajuda02 & var_ajuda03



    'adiciona a fraccao
    var_ajuda01 = txt_fraccao.Text
    Do While (Len(var_ajuda01) < 3)
       var_ajuda01 = "0" & var_ajuda01
    Loop
    criacao_trama = criacao_trama & var_ajuda01



    'adiciona o power down
    var_ajuda01 = Int(txt_power_down.Text / 256)
    var_ajuda02 = txt_power_down.Text - (Int(txt_power_down.Text / 256) * 256)
    Do While (Len(var_ajuda01) < 3)
       var_ajuda01 = "0" & var_ajuda01
    Loop
    Do While (Len(var_ajuda02) < 3)
       var_ajuda02 = "0" & var_ajuda02
    Loop
    criacao_trama = criacao_trama & var_ajuda01 & var_ajuda02



    'adiciona a tarifa
    txt_tarifa01.Text = Int(txt_tarifa01.Text) * 100
    var_ajuda01 = Int(txt_tarifa01.Text / 256)
    var_ajuda02 = txt_tarifa01.Text - (Int(txt_tarifa01.Text / 256) * 256)
    Do While (Len(var_ajuda01) < 3)
       var_ajuda01 = "0" & var_ajuda01
    Loop
    Do While (Len(var_ajuda02) < 3)
       var_ajuda02 = "0" & var_ajuda02
    Loop
    criacao_trama = criacao_trama & var_ajuda01 & var_ajuda02
    txt_tarifa01.Text = Int(txt_tarifa01.Text) / 100
    txt_tarifa01.Text = FormatCurrency(txt_tarifa01.Text, 2)

    'adiciona o total facturado
    txt_total_facturado.Text = Int(txt_total_facturado.Text) * 100
    var_ajuda01 = Int(txt_total_facturado.Text / 65536)
    var_ajuda02 = Int((txt_total_facturado.Text - (Int(txt_total_facturado.Text / 65536) * 65536)) / 256)
    var_ajuda03 = (txt_total_facturado.Text - (Int(txt_total_facturado.Text / 65536) * 65536)) -
(Int((txt_total_facturado.Text - (Int(txt_total_facturado.Text / 65536) * 65536)) / 256) * 256)
    Do While (Len(var_ajuda01) < 3)
       var_ajuda01 = "0" & var_ajuda01
    Loop
    Do While (Len(var_ajuda02) < 3)
       var_ajuda02 = "0" & var_ajuda02
    Loop
    Do While (Len(var_ajuda03) < 3)
       var_ajuda03 = "0" & var_ajuda03
    Loop
    criacao_trama = criacao_trama & var_ajuda01 & var_ajuda02 & var_ajuda03
    txt_total_facturado.Text = Int(txt_total_facturado.Text) / 100
    txt_total_facturado.Text = FormatCurrency(txt_total_facturado.Text, 2)


    'adiciona a password
```

```
    var_ajuda01 = Int(txt_pass.Text / 65536)
    var_ajuda02 = Int((txt_pass.Text - (Int(txt_pass.Text / 65536) * 65536)) / 256)
    var_ajuda03 = (txt_pass.Text - (Int(txt_pass.Text / 65536) * 65536)) - (Int((txt_pass.Text - (Int(txt_pass.Text /
65536) * 65536)) / 256) * 256)
    Do While (Len(var_ajuda01) < 3)
        var_ajuda01 = "0" & var_ajuda01
    Loop
    Do While (Len(var_ajuda02) < 3)
        var_ajuda02 = "0" & var_ajuda02
    Loop
    Do While (Len(var_ajuda03) < 3)
        var_ajuda03 = "0" & var_ajuda03
    Loop
    trama_envio_escrita1.Text = criacao_trama & var_ajuda01 & var_ajuda02 & var_ajuda03 & "#$"

    'envia a trama
    MSComm1.Output = trama_envio_escrita1.Text
Else
    MsgBox ("Favor preencher todos os campos")
End If
End Sub

‘comando sair
Private Sub Command7_Click()
End
End Sub


Private Sub Form_Load()
' Fire Rx Event Every Two Bytes
MSComm1.RThreshold = 1

' When Inputting Data, Input 2 Bytes at a time
MSComm1.InputLen = 1

' 2400 Baud, No Parity, 8 Data Bits, 1 Stop Bit
MSComm1.Settings = "19200,N,8,1"

' Disable DTR
MSComm1.DTREnable = False

' Open COM1
MSComm1.CommPort = 1
MSComm1.PortOpen = True



'verifica quais estão ligados ou não
For i = 1 To 10
    If (txt_config_sn(i) <> "-") Then
        imagem_mesa(i).Picture = LoadPicture("e:\dissertação\vb\bilhar_fechado.jpg")
    Else
        imagem_mesa(i).Picture = LoadPicture("e:\dissertação\vb\bilhar_inactivo.jpg")
    End If

Next i
End Sub

Private Sub label_pagar_Click(Index As Integer)
tt_a_pagar2(Index).Visible = False
label_pagar(Index).Visible = False
End Sub

Private Sub MSComm1_OnComm()
Dim sData As String ' Holds our incoming data
Dim lWord As Long       ' Holds the Word result

' If comEvReceive Event then get data and display
If MSComm1.CommEvent = comEvReceive Then
    sData = MSComm1.Input ' Get data (2 bytes)


    ' Convert value to a string and display
    Text1.Text = Text1.Text & sData
```

```
    If sData = "%" Then
        txt_recebido.Text = trama_recebida
        trama_recebida = ""
    Else
        trama_recebida = trama_recebida & sData
    End If

End If
End Sub


Private Sub Form_Unload(Cancel As Integer)
MSComm1.PortOpen = False
End Sub


Private Sub TabStrip1_Click()

End Sub




Private Sub SSTab1_DblClick()

End Sub



Private Sub tt_a_pagar2_Click(Index As Integer)
tt_a_pagar2(Index).Visible = False
label_pagar(Index).Visible = False
End Sub

Private Sub txt_recebido_Change()

trama_alterada = ""

'caso esteja a tratar uma trama de rede
If Right(Left(txt_recebido.Text, 5), 2) = "$R" Then

    trama_alterada = Right(txt_recebido, Len(txt_recebido) - 5)         'fica: SN0X5X1X264#$

    'verifica o serial number
    If Left(trama_alterada, 2) = "SN" Then
        trama_alterada = Right(trama_alterada, Len(trama_alterada) - 2)         'fica: 0X5X2X312#$
        'verifica o ano
        trama_sn_ano = ""
        Do While Left(trama_alterada, 1) <> "X"
            trama_sn_ano = trama_sn_ano + Left(trama_alterada, 1)   'guarda variavel ano
            trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)     'fica: 5X1X357#$
        Loop
        If (Len(trama_sn_ano) < 2) Then
            trama_sn_ano = "0" + trama_sn_ano
        End If
        trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)
        'verifica o mês
        trama_sn_mes = ""
        Do While Left(trama_alterada, 1) <> "X"
            trama_sn_mes = trama_sn_mes + Left(trama_alterada, 1) 'guarda variavel mes
            trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)     'fica: 1X357#$
        Loop
        If (Len(trama_sn_mes) < 2) Then
            trama_sn_mes = "0" + trama_sn_mes
        End If
        trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)
        'verifica o numero
        trama_sn_num = ""
        Do While Left(trama_alterada, 1) <> "X"
            trama_sn_num = trama_sn_num + Left(trama_alterada, 1) 'guarda variavel numero
            trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)     'fica: 357#$
        Loop
        If (Len(trama_sn_num) < 2) Then
            trama_sn_num = "0" + trama_sn_num
        End If
```

```
      trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)

      If Left(trama_alterada, 2) = "FG" Then
         For i = 1 To 10
            If (txt_config_sn(i).Text = trama_sn_completo) Then
               'mete a imagem em jogo
               imagem_mesa(i).Picture = LoadPicture("e:\dissertação\vb\bilhar_fechado.jpg")
               tt_a_pagar2(i).Text = tt_a_pagar(i).Text
               tt_a_pagar(i).Visible = False
               tt_a_pagar2(i).Visible = True
               label_pagar(i).Visible = True
            End If
         Next i

      'não está a fechar a gaveta
      Else
         'verifica o valor a pagar
         trama_val_a_pagar = ""
         Do While Left(trama_alterada, 1) <> "#"
            trama_val_a_pagar = trama_val_a_pagar + Left(trama_alterada, 1) 'guarda variavel numero
            trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)     'fica: $
         Loop
         trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)


         'verifica se a trama é bem terminada
         If (Left(trama_alterada, 1) = "$") Then
            'aqui vai gravar os dados
            trama_sn_completo = trama_sn_ano + trama_sn_mes + "-" + trama_sn_num
            trama_val_a_pagar = Int(trama_val_a_pagar) / 100
            'adiciona o valor ao menu rede
            For i = 1 To 10
               If (txt_config_sn(i).Text = trama_sn_completo) Then
                  tt_a_pagar(i).Text = trama_val_a_pagar
                  tt_a_pagar(i).Text = FormatCurrency(tt_a_pagar(i).Text, 2)

                  'mete a imagem em jogo
                  imagem_mesa(i).Picture = LoadPicture("e:\dissertação\vb\bilhar_aberto.jpg")

                  'certifica-se que a parte do pagamento n aparece
                  tt_a_pagar2(i).Visible = False
                  tt_a_pagar(i).Visible = True
                  label_pagar(i).Visible = False
               End If
            Next i
         End If
      End If
   End If
End If

'caso esteja a tratar de uma trama de leitura individual
If Right(Left(txt_recebido.Text, 5), 2) = "$#" Then
   trama_alterada = Right(txt_recebido, Len(txt_recebido) - 5)                    'fica:
SN0X5X2XF3XPD31XT01600XTF950994XPW12345#$%

   'verifica o serial number
   If Left(trama_alterada, 2) = "SN" Then
      trama_alterada = Right(trama_alterada, Len(trama_alterada) - 2)             'fica:
0X5X2XF3XPD31XT01600XTF950994XPW12345#$%
      'verifica o ano
      trama_sn_ano = ""
      Do While Left(trama_alterada, 1) <> "X"
         trama_sn_ano = trama_sn_ano + Left(trama_alterada, 1)   'guarda variavel ano
         trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)          'fica:
5X2XF3XPD31XT01600XTF950994XPW12345#$%
      Loop
      If (Len(trama_sn_ano) < 2) Then
         trama_sn_ano = "0" + trama_sn_ano
      End If
      trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)
      'verifica o mês
      trama_sn_mes = ""
      Do While Left(trama_alterada, 1) <> "X"
         trama_sn_mes = trama_sn_mes + Left(trama_alterada, 1) 'guarda variavel mes
```

```
        trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)                    'fica:
2XF3XPD31XT01600XTF950994XPW12345#$%
    Loop
    If (Len(trama_sn_mes) < 2) Then
        trama_sn_mes = "0" + trama_sn_mes
    End If
    trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)
    'verifica o numero
    trama_sn_num = ""
    Do While Left(trama_alterada, 1) <> "X"
        trama_sn_num = trama_sn_num + Left(trama_alterada, 1) 'guarda variavel numero
        trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)                    'fica:
F3XPD31XT01600XTF950994XPW12345#$%
    Loop
    If (Len(trama_sn_num) < 2) Then
        trama_sn_num = "0" + trama_sn_num
    End If
    trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)


    'verifica a fraccao
    If (Left(trama_alterada, 1) = "F") Then
        trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)                    'fica:
3XPD31XT01600XTF950994XPW12345#$%
        trama_fraccao = ""
        Do While Left(trama_alterada, 1) <> "X"
            trama_fraccao = trama_fraccao + Left(trama_alterada, 1) 'guarda variavel numero
            trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)
        Loop
        trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)                    'fica:
PD31XT01600XTF950994XPW12345#$%


        'verifica o powerdown
        If (Left(trama_alterada, 2) = "PD") Then
            trama_alterada = Right(trama_alterada, Len(trama_alterada) - 2)
            trama_power_down = ""
            Do While Left(trama_alterada, 1) <> "X"
                trama_power_down = trama_power_down + Left(trama_alterada, 1) 'guarda variavel numero
                trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)
            Loop
            trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)    'fica: T01600XTF950994XPW12345#$%


            'verifica a tarifa 01
            If (Left(trama_alterada, 3) = "T01") Then
                trama_alterada = Right(trama_alterada, Len(trama_alterada) - 3)
                trama_tarifa01 = ""
                Do While Left(trama_alterada, 1) <> "X"
                    trama_tarifa01 = trama_tarifa01 + Left(trama_alterada, 1) 'guarda variavel numero
                    trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)
                Loop
                trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)    'fica: TF950994XPW12345#$%


                'verifica o total facturado
                If (Left(trama_alterada, 2) = "TF") Then
                    trama_alterada = Right(trama_alterada, Len(trama_alterada) - 2)
                    trama_total_facturado = ""
                    Do While Left(trama_alterada, 1) <> "X"
                        trama_total_facturado = trama_total_facturado + Left(trama_alterada, 1) 'guarda variavel numero
                        trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)
                    Loop
                    trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)    'fica: PW12345#$%


                    'verifica a password
                    If (Left(trama_alterada, 2) = "PW") Then
                        trama_alterada = Right(trama_alterada, Len(trama_alterada) - 2)
                        trama_password = ""
                        Do While Left(trama_alterada, 1) <> "#"
                            trama_password = trama_password + Left(trama_alterada, 1) 'guarda variavel numero
                            trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)
                        Loop
                        trama_alterada = Right(trama_alterada, Len(trama_alterada) - 1)    'fica: $
```

```
                    'verifica se a trama é bem terminada
                    If (Left(trama_alterada, 1) = "$") Then
                        'aqui vai gravar os dados
                        txt_sn.Text = trama_sn_ano + trama_sn_mes + "-" + trama_sn_num
                        txt_fraccao.Text = trama_fraccao
                        txt_power_down.Text = trama_power_down
                        txt_tarifa01.Text = Int(trama_tarifa01) / 100
                        txt_tarifa01.Text = FormatCurrency(txt_tarifa01.Text, 2)
                        txt_total_facturado.Text = Int(trama_total_facturado) / 100
                        txt_total_facturado.Text = FormatCurrency(txt_total_facturado.Text, 2)
                        txt_pass.Text = trama_password

                    End If
                End If
            End If
        End If
    End If
  End If
End If

End Sub
```