

UNIVERSIDADE LUTERANA DO BRASIL

PRÓ-REITORIA DE GRADUAÇÃO

CURSO DE ENGENHARIA ELÉTRICA

DESENVOLVIMENTO DE UM SISTEMA DE

AQUISIÇÃO DE MÚLTIPLOS CANAIS

COM INTERFACE ZIGBEE

DIOGO KOENIG

CANOAS, JUNHO DE 2007

DIOGO KOENIG

**DESENVOLVIMENTO DE UM SISTEMA DE
AQUISIÇÃO DE MÚLTIPLOS CANAIS
COM INTERFACE ZIGBEE**

Trabalho de Conclusão de Curso
apresentado como requisito para a
obtenção do grau de Engenheiro
Eletricista pela Universidade Luterana do
Brasil – Campus Canoas – Curso de
Engenharia Elétrica.

Orientador: Prof. Dr. Alexandre Balbinot

Canoas, Junho de 2007

DIOGO KOENIG

**DESENVOLVIMENTO DE UM SISTEMA DE
AQUISIÇÃO DE MÚLTIPLOS CANAIS
COM INTERFACE ZIGBEE**

BANCA AVALIADORA:

Prof. Dr. Valner João Brusamarello _____

Prof. M.E. Paulo César Cardoso Godoy _____

ORIENTADOR

Prof. Dr. Alexandre Balbinot _____

Trabalho apresentado e aprovado em:

Agradecimentos

Aos familiares e amigos pelo
constante apoio.

RESUMO

O presente trabalho apresenta o desenvolvimento de um sistema de aquisição de dados, em plataforma microcontrolada, de oito canais de conversão analógico-digital com resolução de 16 *bits*, e integrado a uma rede com protocolo Zigbee. São apresentados os principais conceitos do protocolo ZigBee, as características das topologias de redes *wireless* suportadas e as formas de comunicação. Alguns conceitos sobre a boa prática no desenvolvimento de placas de circuito impresso, planos de terra analógicos e digitais e ruído, são abordados durante o desenvolvimento do texto. O *software* desenvolvido, em conjunto para o sistema implementado, foi avaliado para aquisição de dados entre taxas de uma amostra por segundo e 5000 amostras por segundo.

Palavras chaves: Zigbee, wireless, aquisição de dados, conversão A/D.

ABSTRACT

This work proposes the development of a data acquisition system, on a embedded system, witch has eight channel 16 bit resolution analog-to-digital conversion, integrated with a zigbee protocol network. The Zigbee protocol's main concepts are presented, as well as the features of the supported wireless network topologies and its communication ways. Some concepts about the good pratics on the development of printed circuit boards, analog and digital ground plans and noise are considered during the text disclosure. The developed software for the implemented system, was evaluated for data acquisition of 1 to 5000 samples per second rates.

Keywords: Zigbee, wireless, data aquisition, A/D conversion.

SUMÁRIO

| | |
|---|----|
| RESUMO | 5 |
| ABSTRACT | 6 |
| LISTA DE FIGURAS | 10 |
| LISTA DE TABELAS | 12 |
| INTRODUÇÃO..... | 13 |
| 1 Protocolo zigbee | 15 |
| 1.2.1 IEEE 802.15.4..... | 16 |
| 1.2.1. Tipos de Dispositivo IEEE 802.15.4 | 18 |
| 1.2.2 Tipos de dispositivo do protocolo ZigBee..... | 18 |
| 1.2.2.1 Coordenador ZigBee..... | 19 |
| 1.2.2.2 Roteador..... | 19 |
| 1.2.2.3 Dispositivo Final..... | 20 |
| 1.2.3 Topologias de Rede | 20 |
| 1.2.3.1 Topologia Estrela..... | 21 |

| | |
|---|----|
| 1.2.3.2 Topologia Cluster Tree | 21 |
| 1.2.2.3 Topologia Mesh | 22 |
| 1.2.3 Definições do protocolo ZigBee | 26 |
| 1.2.3.1 Tipos de mensagem e binding | 28 |
| 1.2.3.2 Formato de mensagem do protocolo ZigBee | 29 |
| 1.2.3.3 Formato Frame do Protocolo ZigBee | 29 |
| 1.2.3.3.1 Frame KVP | 30 |
| 1.2.3.3.2 Frame MSG | 30 |
| 1.2.3.4 Endereçamento | 31 |
| 1.2.3.4.1 EUI-64 | 31 |
| 1.2.3.4.2 Endereço da Rede | 31 |
| 1.2.3.4.3 Unicast | 31 |
| 1.2.3.4.4 Broadcast | 32 |
| 1.2.3.5 Mecanismos de Transmissão | 32 |
| 1.2.3.6 Roteamento | 33 |
| 1.2.3.7 Associação a Rede | 34 |
| 2 PROCEDIMENTO EXPERIMENTAL | 36 |
| 2.1 Visão geral do projeto | 36 |
| 2.2 Descrição do Hardware | 38 |
| 2.2.1 Fontes | 38 |
| 2.2.2 Memória SRAM | 40 |
| 2.2.3 Conversores Analógicos / Digitais | 41 |
| 2.2.4 Interfaces de Comunicação USB e RS-232 | 43 |

| | |
|--|----|
| 2.2.5 Módulo ZigBee | 43 |
| 2.2.6 Microcontrolador | 45 |
| 2.2.7 Placa de Circuito Impresso | 46 |
| 2.2.8 Topologia de Hardware Implementada..... | 47 |
| 2.3 Descrição do Software | 51 |
| 2.3.1 Configurações realizadas nos Módulos ZigBee | 51 |
| 2.3.1.1. Configurações do Módulo ZigBee na Placa de Recepção..... | 52 |
| 2.3.1.2 Configurações do Módulo ZigBee na Placa de Aquisição | 53 |
| 2.3.2 Software da Placa de Recepção | 54 |
| 2.3.2.1 Execução da Função Principal | 55 |
| 2.3.2.2 Descrição das Funções Implementadas | 60 |
| 2.3.2 Software da Placa de Aquisição | 65 |
| 2.3.2.1 Execução da Função Principal | 66 |
| 2.3.2.2 Descrição das Funções Implementadas | 72 |
| 3 DISCUSSÃO DOS RESULTADOS | 81 |
| 3.1 Projeto de Hardware | 81 |
| 3.1.1 Fontes..... | 82 |
| 3.1.2 Conversores ADCs | 84 |
| 3.1.3 Módulo ZigBee | 90 |
| 3.2 Projeto de Software..... | 90 |
| CONCLUSÕES | 93 |
| REFERÊNCIAS BIBLIOGRÁFICAS | 95 |
| ANEXOS | 98 |

LISTA DE FIGURAS

| | |
|---|----|
| FIGURA 1 - ARQUITETURA ZIGBEE | 16 |
| FIGURA 2 - ESQUEMA DA TOPOLOGIA DE REDE ESTRELA | 21 |
| FIGURA 3 - ESQUEMA DA TOPOLOGIA CLUSTER TREE..... | 22 |
| FIGURA 4 - ESQUEMA DA TOPOLOGIA DE REDE MESH | 23 |
| FIGURA 5 - EXEMPLO DE FUNCIONAMENTO DE UMA REDE ZIGBEE TOPOLOGIA MESH | 24 |
| FIGURA 6 - EXEMPLO DE FUNCIONAMENTO DE UMA REDE ZIGBEE TOPOLOGIA MESH | 25 |
| FIGURA 7 - EXEMPLO DE FUNCIONAMENTO DE UMA REDE ZIGBEE TOPOLOGIA MESH | 25 |
| FIGURA 8 - EXEMPLO DE FUNCIONAMENTO DE UMA REDE ZIGBEE TOPOLOGIA MESH | 26 |
| FIGURA 9 - EXEMPLO DE ARQUITETURA DE PERFIL DO PROTOCOLO ZIGBEE | 27 |
| FIGURA 10 - EXEMPLO DE BINDING E TABELA DE VINCULAÇÃO | 28 |
| FIGURA 11 - SISTEMA IMPLEMENTADO | 37 |
| FIGURA 12 - FLUXOGRAMA DO SOFTWARE PRINCIPAL DA PLACA DE RECEPÇÃO | 56 |
| FIGURA 13 - FLUXOGRAMA DO SOFTWARE PRINCIPAL DA PLACA DE RECEPÇÃO | 57 |
| FIGURA 14 - FLUXOGRAMA DO SOFTWARE PRINCIPAL DA PLACA DE RECEPÇÃO | 58 |
| FIGURA 15 - FLUXOGRAMA DO SOFTWARE PRINCIPAL DA PLACA DE AQUISIÇÃO | 67 |
| FIGURA 16 - FLUXOGRAMA DO SOFTWARE PRINCIPAL DA PLACA DE AQUISIÇÃO | 68 |
| FIGURA 17 - FLUXOGRAMA DO SOFTWARE PRINCIPAL DA PLACA DE AQUISIÇÃO | 69 |
| FIGURA 18 - HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 0 | 86 |

| | |
|--|----|
| FIGURA 19 - HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 1 | 86 |
| FIGURA 20 - HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 2 | 87 |
| FIGURA 21 - HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 3 | 87 |
| FIGURA 22 - HISTOGRAMA PARA TENSÃO COTÍNUA NO CANAL 4..... | 88 |
| FIGURA 23 - HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 5 | 88 |
| FIGURA 24 - HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 6..... | 89 |
| FIGURA 25 - HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 7 | 89 |

LISTA DE TABELAS

| | |
|--|----|
| TABELA 1 – BANDAS DE FREQUÊNCIA E TAXAS DE TRANSFERÊNCIA | 17 |
| TABELA 2 – RELAÇÃO ENTRE DISPOSITIVOS IEEE E PROTOCOLO ZIGBEE | 18 |
| TABELA 3 – TABELA VERDADE PARA ACIONAMENTO DOS BARRAMENTOS DE DADOS..... | 49 |

INTRODUÇÃO

Através do avanço da tecnologia na área de telecomunicações, novos padrões de comunicação vêm surgindo para atender mercados emergentes. Redes *wireless* estão cada vez mais presentes no dia a dia de usuários domésticos, comerciais e industriais. Com o aumento da demanda, custos para aquisição de equipamentos destinados a esta área estão sendo reduzidos.

Inúmeras aplicações não necessitam de grandes volume de dados, ou altas taxas de transferência, entretanto necessitam de muitos nós de comunicação. A IEEE 802.15.4 veio para atender esta necessidade, estipulando as diretrizes para redes pessoais com grande quantidade de dispositivos e baixas taxas de transferência, em bandas de frequências livres para utilização pública. Este padrão acaba por chamar a atenção do mercado e, tomado como base, surge o protocolo de rede ZigBee.

Muitos dispositivos vêm sendo produzidos no mercado utilizando este novo protocolo, e módulos de comunicação, com interfaces RS-232, SPI, I2C e inúmeros outros periféricos, encapsulados em tamanho reduzido, tornam-se bastante atrativos para utilização nas mais diversas áreas. Sendo a telemetria uma área em expansão, esta tecnologia vem como mais uma alternativa para a realização de links de rádio.

Muitas medições e ensaios necessitam de inúmeros canais de aquisição simultâneos.

Em determinadas situações, não é possível contar com um computador e placas de aquisição, devido à natureza do ensaio e tipo de grandeza, que pode danificar equipamentos sensíveis. O presente trabalho visa desenvolver um sistema de aquisição que atenda essas necessidades e que possa ser integrado numa rede *wireless* com protocolo ZigBee.

O sistema a ser apresentado será composto por uma placa de aquisição de dados e uma placa de recepção. A placa de aquisição será portátil, com oito canais de conversão AD, resolução de 16 *bits* e entrada analógica na faixa de 0V a 5V. Poderão ser amostrados até quatro canais simultaneamente entre taxas de uma amostra por segundo até 5000 amostras por segundo. Um *buffer* de memória também estará disponível para o armazenamento temporário dos dados capturados durante os ensaios. A placa de recepção terá duas interfaces de comunicação disponíveis, USB e RS-232, para que possa ser conectada a um terminal computadorizado. Em ambas as placas serão implementados módulos de comunicação Zigbee para comunicação *wireless*.

No capítulo 1, serão apresentados os principais conceitos do protocolo ZigBee, que são fundamentais para a entender e implementar a rede *wireless*. O capítulo 2 apresentará o desenvolvimento do projeto em si, justificando a escolha e tomada de decisão para cada bloco de *hardware* e *software*. No capítulo 3 serão apresentados os resultados que foram obtidos nos testes de validação para cada bloco do projeto.

1 PROTOCOLO ZIGBEE

O termo ZigBee é usado para descrever um protocolo para redes pessoais *wireless*. (WPAN). O protocolo é de propriedade da *ZigBee Alliance*, uma sociedade de mais de 70 empresas que têm trabalhado para criar e promover um padrão capaz de possibilitar um controle seguro, de baixo custo e de baixa potência em redes sem fio para o controle de diversos equipamentos, incluindo soluções para a automação predial, aplicações em telemedicina e entretenimento.

A razão para promover o novo protocolo como um padrão, é de assegurar a interoperabilidade entre dispositivos produzidos por diferentes fabricantes.

O *hardware* e *software* utilizados pelo padrão ZigBee são baseados na norma IEEE 802.15.4 (que foi recentemente revisada), definindo estes em termos dos *layers* PHY e MAC.

A *ZigBee Alliance* adicionou às especificações do IEEE 802.15.4 os *layers* NWK, APL, e um terceiro *layer* de segurança, para completar o que é chamado de *ZigBee stack*, representado esquematicamente na figura 1.

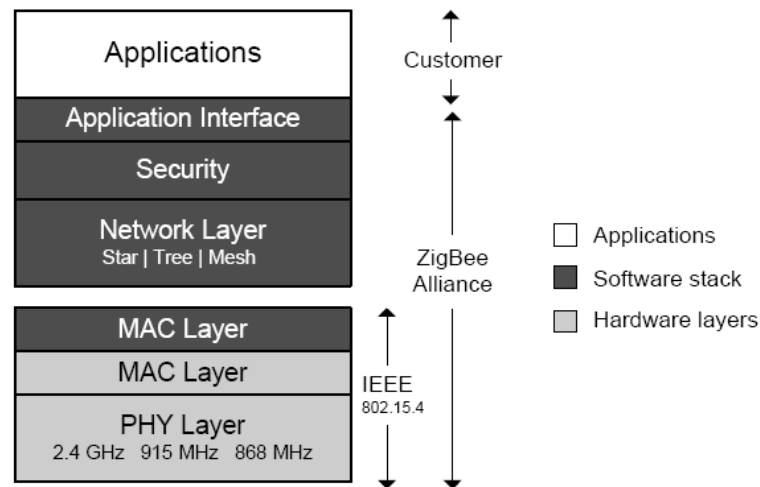


FIGURA 1 - ARQUITETURA ZIGBEE

FONTE: EMBER CORPORATION

1.2.1 IEEE 802.15.4

O protocolo ZigBee utiliza a especificação IEEE 802.15.4 [2] como seus *layers* MAC e PHY. O IEEE 802.15.4 define três frequências de operação: 2,4GHz, 915MHz e 868MHz. Cada banda de frequência oferece um número fixo de canais. Para a frequência de 2,4GHz são disponibilizados 16 canais, para 915MHz 10 canais e para 868MHz apenas 1 canal. As frequências centrais (f_c) de cada canal e sua respectiva banda de frequência são dadas pelas seguintes expressões:

$$f_c = 868,3 * 10^6, \text{ para } k = 0;$$

$$f_c = 906,10^6 * (k - 1), \text{ para } k = 1, 2, \dots, 10;$$

$$f_c = 2405,10^6 + 5*(k - 11), \text{ para } k = 11, 12, \dots, 26;$$

Onde k é o número do canal.

A taxa de transferência do protocolo depende da banda de frequência em operação. A banda de 2,4GHz provê taxa de 250kbps, a banda de 915MHz provê taxa de 40kbps e a banda de 868MHz provê taxa de 20kbps. A taxa de dados efetiva será menor que a especificada, devido aos cabeçalhos dos pacotes, processamentos e *delays*. A tabela 1

representa as informações discutidas, retiradas diretamente do documento da especificação ZigBee.

TABELA 1 – BANDAS DE FREQUÊNCIA E TAXAS DE TRANSFERÊNCIA – FONTE: IEEE STANDARDS

| PHY (MHz) | Frequency band (MHz) | Spreading parameters | | Data parameters | | |
|--------------|----------------------------|------------------------|------------|--------------------|----------------------------|----------------------|
| | | Chip rate (kchip/s) | Modulation | Bit rate (kb/s) | Symbol rate (ksymbol/s) | Symbols |
| 868/915 | 868–868.6 | 300 | BPSK | 20 | 20 | Binary |
| | 902–928 | 600 | BPSK | 40 | 40 | Binary |
| 2450 | 2400–2483.5 | 2000 | O-QPSK | 250 | 62.5 | 16-ary Orthogonal |

Segundo a IEEE 802.15.4, o tamanho máximo de um pacote MAC é de 127 bytes, incluindo um valor de 16 bits para o CRC. Os 16 bits de CRC verificam a integridade do *frame*. Além do mais, o IEEE 802.15.4 utiliza, opcionalmente, mecanismos de confirmação para a transferência de dados. Com este método, todos os *frames* com o *flag* ACK habilitado, são confirmados pelo receptor através de resposta enviada ao transmissor. Desta maneira há a real confirmação de que o *frame* foi recebido pelo nó de destino. Se o *frame* é transmitido com o *flag* ACK habilitado e a confirmação não é recebida dentro de certo tempo, o transmissor realizará novas transmissões, um certo número de vezes, antes de acusar um erro. É importante ressaltar que a recepção de uma confirmação indica simplesmente que o *frame* foi propriamente recebido pelo *layer* MAC. Não indica, contudo, que o *frame* foi processado corretamente. É possível que o *layer* MAC do nó receptor recebeu e confirmou a entrega, mas devido a alguma deficiência no processamento, o *frame* pode ter sido descartado pelos *layers* superiores. Como consequência, *layers* superiores podem requerer confirmações adicionais.

1.2.1. Tipos de Dispositivo IEEE 802.15.4

A IEEE 802.15.4 define dois tipos de dispositivos em uma rede ZigBee:

Full Function Device (FFD): pode funcionar em toda a topologia do padrão, desempenhando a função de coordenador da rede e conseqüentemente ter acesso a todos os outros dispositivos. Trata-se de dispositivos de construção mais complexa. Possui todos os serviços do protocolo. Apresenta maior consumo de energia devido seu *transceiver* de rádio permanecer ligado mesmo com o dispositivo inoperante.

Reduced Function Device (RFD): é limitado a uma configuração com topologia em estrela, não podendo atuar como um coordenador da rede. São dispositivos de construção mais simples. Não possui todos os serviços do protocolo. É aplicado para realizar atividades extremamente simples que não necessitem de grandes quantidades de dados. Só pode comunicar-se com outros FFDs. Apresenta menor consumo de energia devido seu *transceiver* rádio ser desligado quando o dispositivo está inoperante.

Estes dispositivos são mostrados na tabela 2, que indica como eles são relacionados ao protocolo ZigBee.

TABELA 2 – RELAÇÃO ENTRE DISPOSITIVOS IEEE E PROTOCOLO ZIGBEE

| Dispositivo do Protocolo ZigBee | Tipo de Dispositivo IEEE |
|---------------------------------|--------------------------|
| Coordenador | FFD |
| Roteador | FFD |
| Dispositivo Final | FFD ou RFD |

1.2.2 Tipos de dispositivo do protocolo ZigBee

Os dispositivos FFD e RFD, definidos pela IEEE 802.15.4, podem operar em três modos diferentes no protocolo ZigBee [4]: coordenador, roteador e dispositivo final, descritos a seguir.

1.2.2.1 Coordenador ZigBee

Um coordenador é responsável por iniciar uma rede, sendo que existe exatamente um coordenador ZigBee em cada rede. É obrigatoriamente um dispositivo do tipo FFD. Tem como funções típicas transmitir sinalizadores, administrar nós, armazenar informações sobre a rede e rotear mensagens.

Além disto, o coordenador é capaz de criar uma rede tornando-se raiz da árvore dessa rede, sendo, portanto, o único dispositivo capaz de comutar dados entre redes.

Em redes do tipo estrela, o coordenador é o único dispositivo que controla o funcionamento da rede. Todos os elementos da rede são dispositivos finais e toda a comunicação é realizada pelo coordenador.

Em redes tipo *mesh*, o coordenador é responsável por iniciar a rede e configurar certos parâmetros chaves, mas a rede pode ser estendida pelo uso de roteadores.

O coordenador também define o endereço do dispositivo que servirá como *trust center*. Este Dispositivo é responsável por realizar operações relacionadas com a segurança da rede. Assim como o coordenador, só pode haver um *trust center* por rede, geralmente sendo o próprio coordenador.

1.2.2.2 Roteador

Os dispositivos roteadores agem como um roteador intermediário, roteando dados para outros dispositivos, preservando a rede local, mantendo contato com seus vizinhos próximos (tabela *neighbour*) e, armazenando e transmitindo dados que são de interesse dos seus dispositivos associados (*child*). É obrigatoriamente um dispositivo do tipo FFD.

1.2.2.3 Dispositivo Final

Um dispositivo final pode ser do tipo RFD ou FFD. Não possuem nenhuma responsabilidade de roteamento na rede. Eles associam-se a rede através de roteadores ou coordenadores, também conhecidos como dispositivos *parents*. Os dispositivos finais só podem comunicar-se através de seu dispositivo *parent*, portanto não podem trocar informações entre si ou com outros roteadores ou coordenadores.

O *parent* armazenará mensagens enviadas para seu dispositivo final (*child*) enquanto este estiver inoperante. Para que não haja perda de informações devido à sobrecarga do buffer, o *child* deverá verificar periodicamente, se não há mensagens pendentes junto a seu *parent*. O *parent* também agirá como transmissor das mensagens do seu *child*. Por exemplo, se um *child* deseja enviar uma mensagem para toda a rede, primeiro ele deve enviar a mensagem para seu *parent*, que então transmitirá a mensagem para o restante da rede.

Com requer menos memória, pois não precisa armazenar informações de roteamento, é mais barato que um roteador ou um coordenador ZigBee.

1.2.3 Topologias de Rede

Uma rede com protocolo ZigBee pode ter muitos tipos de configuração. Em todos os tipos de configuração, há ao menos dois componentes principais: um nó coordenador e dispositivos finais.

Um coordenador é uma variação especial de um FFD, que implementa uma ampla variedade de serviços do protocolo ZigBee. Um dispositivo final pode ser um FFD ou um RFD. Um RFD é o menor e mais simples nó do protocolo ZigBee, pois implementa o mínimo de serviços do protocolo. Um terceiro componente opcional é o roteador, que pode estar presente em algumas configurações de rede.

A camada de rede ZigBee suporta três topologias de redes: *star* (estrela), *Cluster Tree* (árvore) e *Mesh* (malha).

1.2.3.1 Topologia Estrela

Uma rede tipo estrela consiste de um nó coordenador e um ou mais dispositivos finais. Em uma rede estrela, todos os dispositivos finais comunicam-se somente com o coordenador. Se um dispositivo final necessita realizar comunicação com outro, os dados são enviados ao coordenador, que então os encaminha para o dispositivo de destino.

Pode-se visualizar este processo na figura 2 , onde, tanto os dispositivos do tipo FFD como do tipo RFD se comunicam apenas com o coordenador.

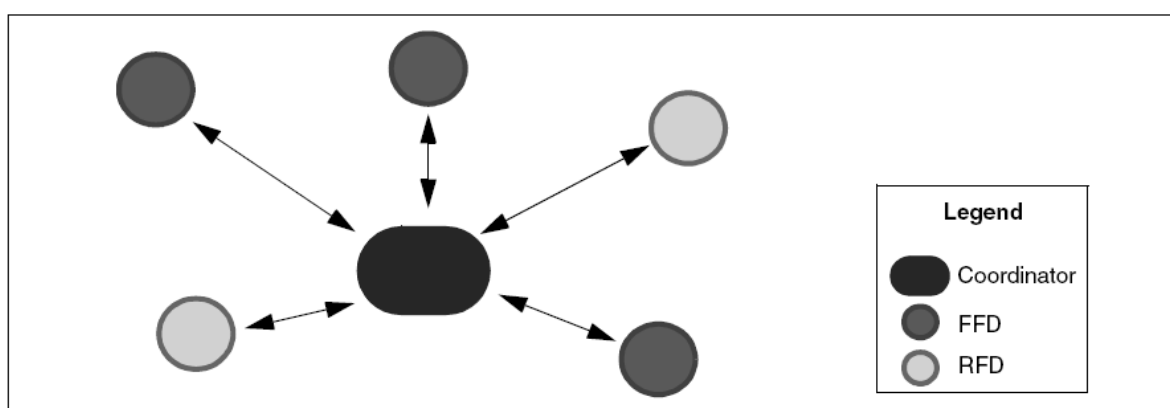


FIGURA 2 - ESQUEMA DA TOPOLOGIA DE REDE ESTRELA

FONTE: MICROCHIP

1.2.3.2 Topologia Cluster Tree

Nesta topologia, dispositivos finais podem associar-se à rede através do coordenador ou roteadores que desempenham duas funções:

- aumentar o número de nós que podem juntar-se à rede;

- aumentar o alcance de atuação da rede.

Dispositivos finais associados a roteadores não devem estar no alcance de rádio do coordenador. Todas as mensagens em uma topologia *cluster tree* são roteadas de acordo com a hierarquia da mesma, exemplificada na figura 3.

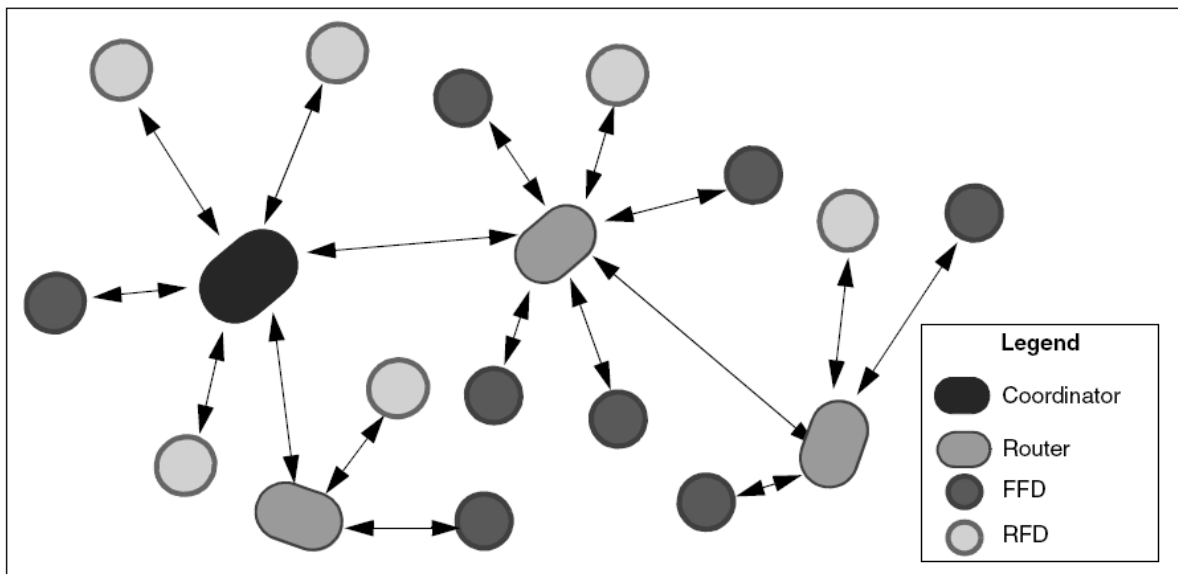


FIGURA 3 - ESQUEMA DA TOPOLOGIA CLUSTER TREE

FONTE: MICROCHIP

1.2.2.3 Topologia Mesh

Uma rede *mesh* (figura 4) é similar a uma rede de topologia *cluster tree*, exceto que FFDs podem rotear mensagens diretamente para outros FFDs, ao invés de seguir a estrutura hierárquica da rede *tree*. Mensagens destinadas a RFDs também são enviadas para o dispositivo *parent*, associado ao RFD em questão (*child*). As topologias *cluster tree* e *mesh* também são conhecidas como redes do tipo *multi-hop*, devido à capacidade de transmitir dados através de inúmeros nós, até que a mensagem seja entregue ao dispositivo de destino.

Redes de topologia estrela são conhecidas como redes do tipo *single-hop*.

Uma rede com protocolo ZigBee é uma rede de multi-acesso, portanto, todos os nós na rede têm o mesmo acesso ao meio de comunicação. Há dois tipos de mecanismos de multi-acesso, sinalizado e não sinalizado.

Em uma rede não sinalizada, todos os nós da rede têm permissão para transmitir a qualquer momento, enquanto o canal estiver inativo.

Em uma rede sinalizada, os nós têm permissão para transmitir em um espaço de tempo predefinido. O coordenador periodicamente transmite um *superframe*, identificado como um frame sinalizador, e todos os nós da rede são sincronizados por este *superframe*. Cada nó é então atribuído, pelo *superframe*, a um espaço de tempo específico, onde este nó pode receber e enviar dados. Um *superframe* também pode conter um espaço de tempo comum, onde todos os nós competem para acessar o canal.

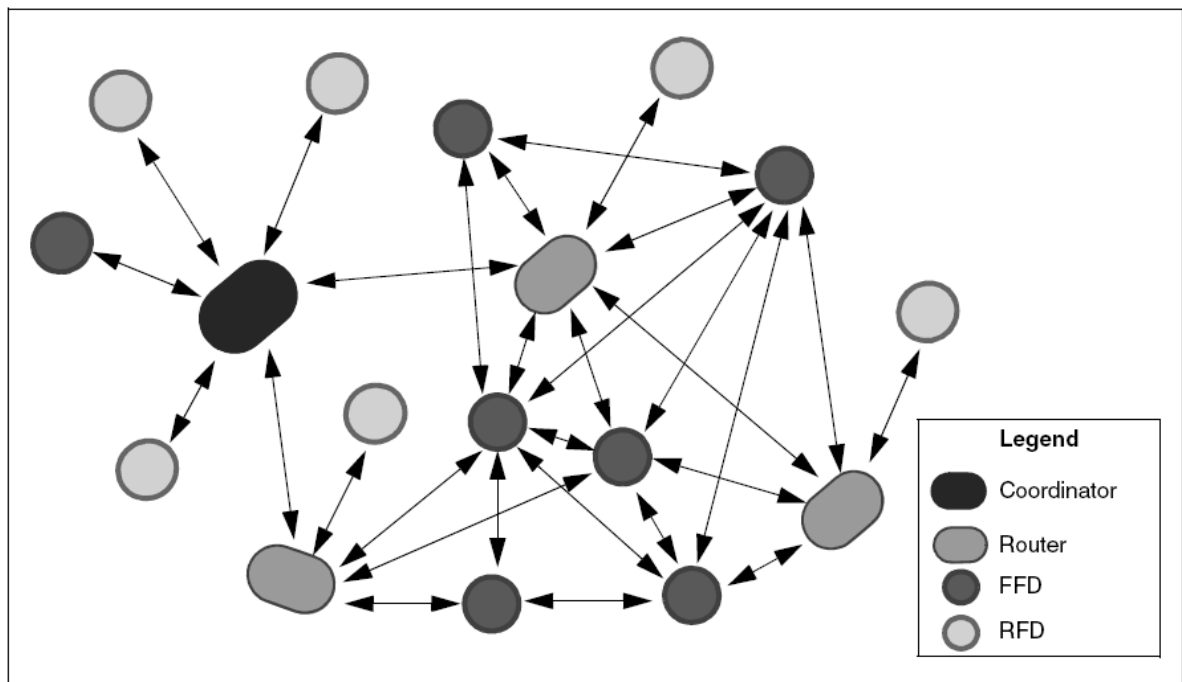


FIGURA 4 - ESQUEMA DA TOPOLOGIA DE REDE MESH

FONTE: MICROCHIP

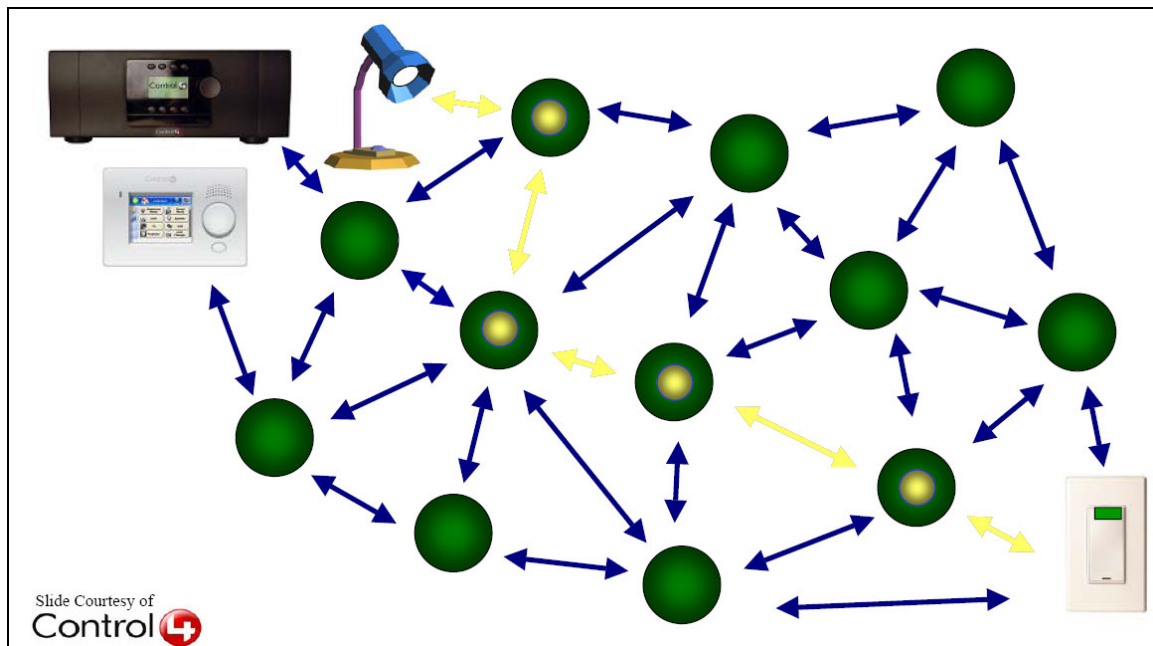


FIGURA 6 - EXEMPLO DE FUNCIONAMENTO DE UMA REDE ZIGBEE TOPOLOGIA MESH

FONTE: BOB HEILE

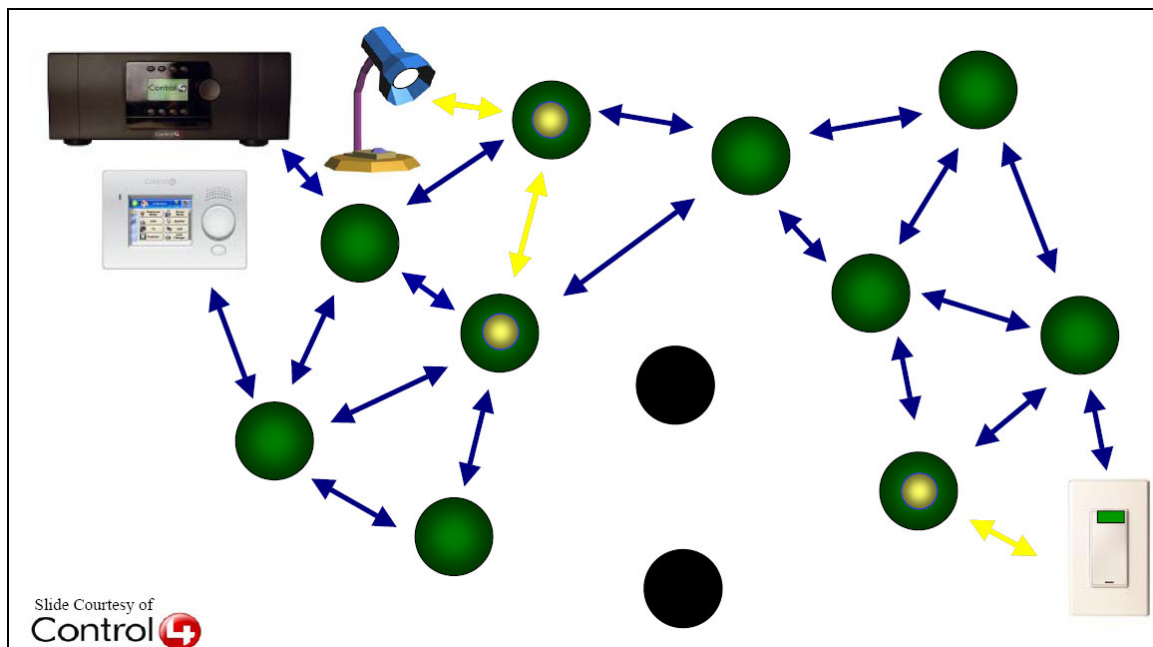


FIGURA 7 - EXEMPLO DE FUNCIONAMENTO DE UMA REDE ZIGBEE TOPOLOGIA MESH

FONTE: BOB HEILE

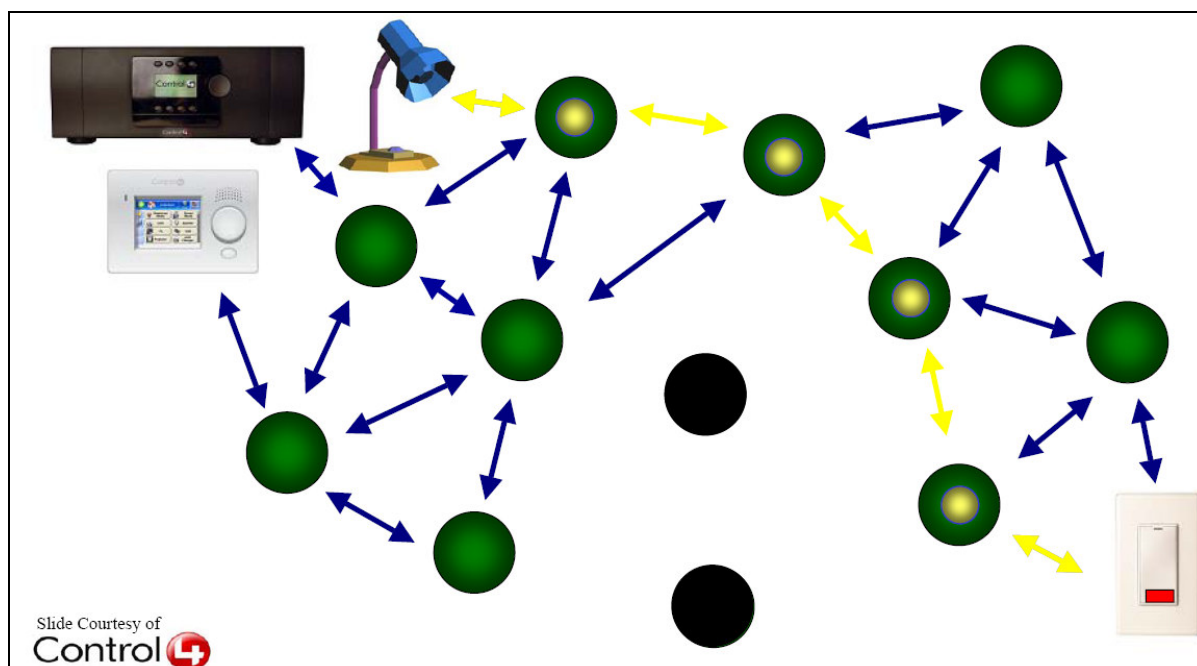


FIGURA 8 - EXEMPLO DE FUNCIONAMENTO DE UMA REDE ZIGBEE TOPOLOGIA MESH

FONTE: BOB HEILE

1.2.3 Definições do protocolo ZigBee

O perfil do protocolo ZigBee é simplesmente a descrição de componentes lógicos (dispositivos) e suas interfaces. Frequentemente não há código associado ao perfil. Cada fragmento de dados que pode ser passado entre dispositivos: como estado de chaves ou leitura de potenciômetros, é chamado de atributo. Cada atributo possui um único identificador. Estes atributos são agrupados em *clusters*. Cada *cluster* também possui um único identificador. Interfaces são especificadas ao nível de *cluster*, não ao nível de atributo, que são transferidos individualmente.

O perfil define os valores das identidades de atributo e *clusters*, assim como o formato de cada atributo. Por exemplo, no perfil de Iluminação para Controle Residencial, o *cluster* OnOffDRC do dispositivo de controle remoto de um dimmer (DRC) contém um

atributo, OnOff, que deve ser um valor não sinalizado de 8 bits, com valor 0xFF significando “ligado”, valor 0x00 significando “desligado” e valor 0xF0 significando “troca de estado”. O perfil também descreve quais *clusters* são obrigatórios e quais são opcionais para cada dispositivo. Ele também pode definir alguns serviços opcionais do protocolo ZigBee como obrigatórios.

O usuário deve escrever seu código utilizando estes conceitos. Ele pode escrever o código da maneira que quiser, agrupando as funções livremente, desde que os *clusters* e serviços obrigatórios sejam suportados por seu código e utilize os atributos que são definidos pelo perfil. Cada bloco funcional de código que suporta um ou mais *clusters* é chamado de *endpoint*. Diferentes dispositivos comunicam-se através de seus *endpoints* e seus *clusters*.

A figura 9 mostra como os vários termos apresentados se relacionam. Ela mostra dois dispositivos do perfil de Iluminação para Controle Residencial. Cada dispositivo tem um único *endpoint*. O controlador de carga (*switch load controller*) possui um *cluster* de entrada no seu *endpoint*. O controlador do interruptor (*switch remote control*) possui um *cluster* de entrada e um *cluster* de saída no seu *endpoint*. O interruptor também poderia ser implementado utilizando-se dois *endpoints*, um para cada *cluster*.

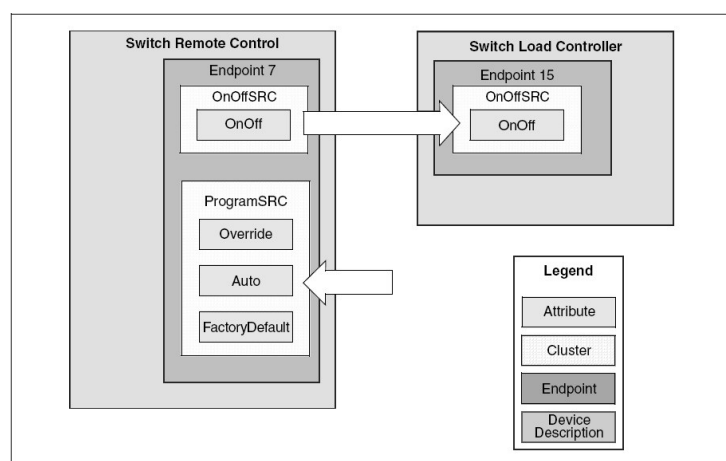


FIGURA 9 - EXEMPLO DE ARQUITETURA DE PERFIL DO PROTOCOLO ZIGBEE

FONTE: MICROCHIP

1.2.3.1 Tipos de mensagem e binding

Dispositivos podem comunicar-se com outros dispositivos na rede se seus endereços são conhecidos. Este tipo de comunicação é realizado de forma direta. Contudo, há uma grande quantidade de cabeçalho envolvido para descobrir e manter estes endereços de destino. O protocolo ZigBee apresenta uma característica chamada *binding* (figura 10), que visa simplificar as comunicações. O coordenador ZigBee pode criar uma tabela de vinculação, a nível de *cluster/endpoint*, entre os serviços e as necessidades dos dispositivos da rede. Cada um desses pares é chamado de *binding*. Um *binding* pode ser solicitado pelos próprios dispositivos, ou pode ser criado pelo coordenador ou outro dispositivo. Uma vez criado um *binding*, dois dispositivos podem comunicar-se através do coordenador. O dispositivo transmissor envia sua mensagem para o coordenador, que então retransmite a mensagem para um ou mais dispositivos de destino. Estas mensagens são chamadas de indiretas.

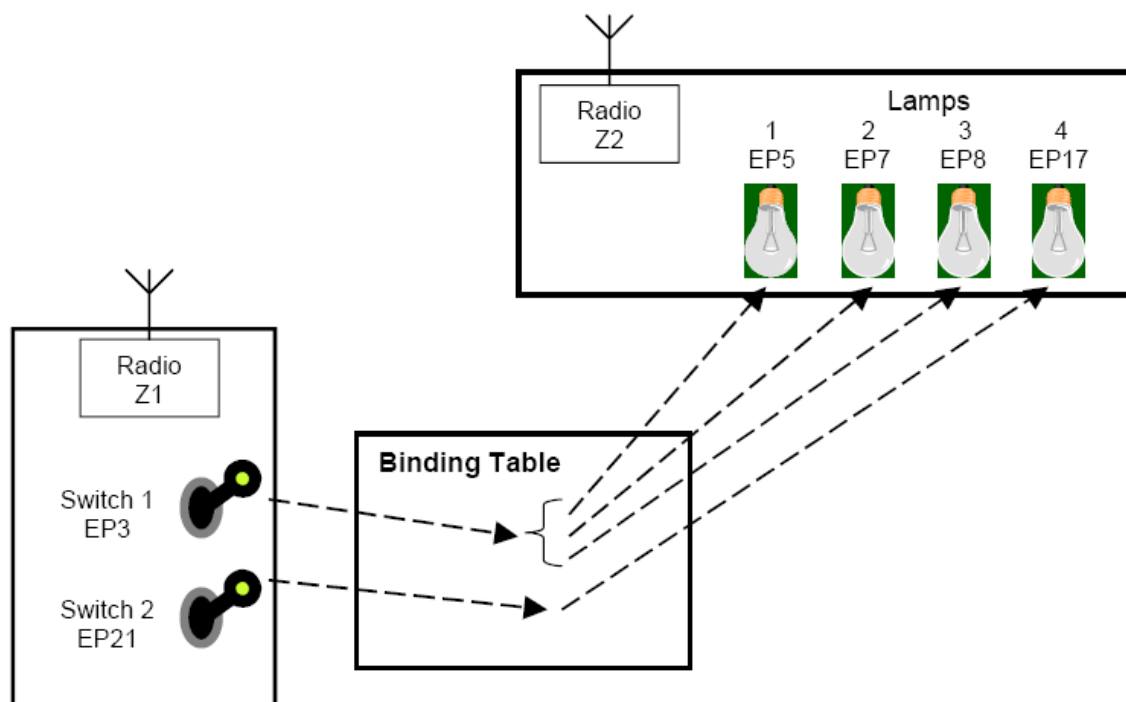


FIGURA 10 - EXEMPLO DE BINDING E TABELA DE VINCULAÇÃO

FONTE: ZIGBEE STANDARDS ORGANIZATION

1.2.3.2 Formato de mensagem do protocolo ZigBee

Uma mensagem do protocolo ZigBee consiste em até 127 bytes dividida nos seguintes campos:

- **cabeçalho MAC**: o cabeçalho do MAC contém campos de controle de *frame*, número sequencial sinalizador (BNS) e informações de endereçamento da mensagem que está sendo transmitida. Note que as informações deste cabeçalho podem não refletir a fonte de origem ou o destino final da mensagem se a mensagem está sendo roteada. A criação e uso deste cabeçalho são transparentes para o código de aplicação.
- **cabeçalho NWK (*Network Layer Header*)**: este cabeçalho contém, entre outras informações, a fonte real e o destino final da mensagem. A criação e uso deste cabeçalho são transparentes ao código de aplicação.
- **cabeçalho APS (*Application Support Sub-Layer Header*)**: este cabeçalho contém a identificação do perfil, identificação do *cluster* e o *endpoint* destino da mensagem atual. A criação e geração deste cabeçalho são transparentes ao código de aplicação.
- **APS Payload**: este campo contém o *frame* do protocolo ZigBee para processamento da aplicação. O código de aplicação é responsável pelo preenchimento deste campo.

1.2.3.3 Formato Frame do Protocolo ZigBee

O protocolo ZigBee define dois formatos de *frame*: KVP (*Key Value Pair*) e MSG (*Message*). Ambos formatos são associados com a identificação de um *cluster*, mas os *frames* KVP são utilizados para transferir informações associadas com um atributo utilizando um formato determinado de estrutura, enquanto *frames* MSG, transferem informações usando uma estrutura mais livre. O perfil da aplicação especificará qual formato de *frame* deverá ser utilizado para transferir cada tipo de dado e o formato de qualquer *frame* MSG. Devido às diferenças entre os formatos de *frames*, um *cluster* pode não utilizar ambos os formatos.

1.2.3.3.1 *Frame KVP*

Um *frame* KVP contém as seguintes informações, nesta ordem:

- 1 – Contagem de operação
- 2 – Tipo de *Frame*
- 3 – Operações
 - Número seqüencial de operação
 - Tipo de comando e tipo de dado do atributo
 - Identificação do atributo
 - Código de erro (opcional)
 - Dado do atributo (tamanho variável)

O “tipo de comando” indica o que a aplicação supostamente deve fazer com a informação. Por exemplo, o comando “*set*”, solicita ao receptor para carregar o valor do atributo, indicado pela “identificação do atributo”, para o valor indicado em “dado do atributo” e, o comando “*get with acknowledge*”, solicita ao receptor para enviar o valor do atributo indicado pela “identificação do atributo”.

1.2.3.3.2 *Frame MSG*

Um *frame* MSG contém as seguintes informações, nesta ordem:

- 1 – Contagem de operação
- 2 – Tipo de *Frame*
- 3 – Operações
 - Número seqüencial da operação
 - Comprimento da operação
 - Dado da operação

Ambos os dispositivos, receptor e transmissor, precisam estar cientes do formato do campo “dado da operação”.

1.2.3.4 Endereçamento

Cada nó da rede ZigBee terá dois endereços: um endereço MAC de 64 bits e em endereço de rede de 16 bits.

1.2.3.4.1 EUI-64

Todo dispositivo que se comunica usando o protocolo ZigBee precisa ter um único endereço MAC de 64 bits. Este endereço é constituído de um campo de 24 bits, chamado OUI (*Organizationally Unique Identifier*), mais um campo de 40 bits determinado pelo fabricante. OUIs são adquiridos junto ao IEEE, para assegurar imparidade global. Para organizações que possuem um OUI utilizado em aplicações ethernet, pode-se utilizar o mesmo para aplicações ZigBee.

1.2.3.4.2 Endereço da Rede

Dispositivos utilizam seus EUI-64 para realizar comunicações enquanto estão associando-se a rede. Após o dispositivo associar-se, sua identificação passa a ser determinada por um endereço de rede de 16 bits, o qual ele usa para realizar qualquer comunicação dentro da rede.

1.2.3.4.3 Unicast

Em mensagem *unicast*, o endereço de destino do nó é fornecido no cabeçalho do

layer MAC. Apenas o dispositivo que tem o endereço indicado receberá a mensagem.

1.2.3.4.4 Broadcast

Num pacote *broadcast*, o endereço de destino no *layer* MAC é 0xFFFF. Qualquer *transceiver* que esteja ligado receberá a mensagem. Esta forma de endereçamento é usada quando o dispositivo está se associando a rede, descobrindo rotas ou executando outras funções de descoberta do protocolo ZigBee. O protocolo implementa uma “confirmação passiva” para pacotes *broadcast*. Isto significa que quando um dispositivo transmite ou retransmite um pacote *broadcast*, ele esperará a retransmissão do pacote por todos os seus vizinhos conhecidos. Se todos os vizinhos não replicarem a mensagem dentro de alguns segundos, ele irá retransmitir o pacote novamente, até que “ouça” a retransmissão de todos vizinhos. Caso as retransmissões dos vizinhos não ocorram após certo número de tentativas, há o estouro do tempo, e transmissões subseqüentes são canceladas.

1.2.3.5 Mecanismos de Transmissão

Em uma rede não sinalizada, quando um dispositivo quer mandar *frames* de dados, ele simplesmente espera que o canal de transmissão fique desocupado. Detectando a condição de canal desocupado, o dispositivo pode transmitir um *frame*.

Se o dispositivo de destino é um FFD, então o *transceiver* está sempre ligado, e outros dispositivos podem transmitir para ele a qualquer momento. Esta capacidade permite a construção de redes *mesh*. Contudo, se o dispositivo é um RFD, então seu *transceiver* pode estar desligado. O RFD não estará apto para receber mensagens enquanto estiver neste estado. Esta condição é tratada através do envio de todas as mensagens que vem e vão do RFD para seu *parent* FFD. Quando o RFD liga novamente seu *transceiver*, ele requisita as mensagens armazenadas pelo seu *parent*. Se o *parent* tem mensagens armazenadas para seu

child, então retransmite-as ao mesmo. Este método permite que RFDs consumam menos energia, mas requer que os FFDs possuam memória RAM suficiente para armazenar mensagens para todos seus dispositivos do tipo *child*. Se o *child* não solicita mensagens durante certo período, as mensagens são então descartadas pelo *parent*.

1.2.3.6 Roteamento

O roteamento é realizado automaticamente pela *Stack*, sem nenhuma intervenção da aplicação final. O roteamento permite que a área da rede seja estendida via associação de dispositivos finais, além do alcance de rádio do coordenador, através de roteadores.

O tipo de roteamento desejado para a mensagem é indicado quando esta é enviada. Há três opções de roteamento:

- ***Suppress***: se há uma rota *mesh* descoberta, a mensagem é roteada através desta. Caso contrário, a mensagem é roteada seguindo a hierarquia de rede *tree*.

- ***Enable***: se uma rota *mesh* existe, a mensagem é enviada ao longo desta. Se a rota *mesh* não está determinada, o FFD pode iniciar uma descoberta de rota. Quando o processo termina, a mensagem é enviada através da rota calculada. Se o FFD não tem capacidade de roteamento, a mensagem será enviada seguindo a hierarquia da rede *tree*.

- ***Force***: se o FFD tem capacidade de roteamento, ele iniciará uma descoberta de rota, mesmo que a rota já exista. Quando o processo é completado, a mensagem é enviada ao longo da rota calculada. Se o FFD não tem capacidade de roteamento, a mensagem será enviada seguindo à hierarquia da rede *tree*. Esta opção deve ser usada com cuidado, pois é gerado grande tráfego na rede. É utilizada principalmente para reparar rotas que foram interrompidas.

1.2.3.7 Associação a Rede

Uma rede Zigbee é estabelecida através de um coordenador. Na inicialização, o coordenador procura por outros coordenadores nos canais que lhe são permitidos. Baseado na energia e número de redes encontradas em cada canal, ele estabelece sua própria rede com uma identificação única de 16 bits, chamada de PAN ID. Tendo a rede sido estabelecida, roteadores e dispositivos finais podem juntar-se a mesma.

Com a rede formada é possível, que devido a mudanças físicas, duas ou mais redes sobreponham-se e um conflito de PAN ID pode surgir. Nesta situação, um coordenador pode iniciar um procedimento de resolução de conflito, onde um dos coordenadores poderá mudar de PAN ID ou canal. O coordenador afetado pela mudança instruirá todos seus dispositivos do tipo *child* para realizar as mudanças necessárias.

Dispositivos ZigBee armazenam informações sobre outros nós da rede, incluindo nós *parent* e *child*, numa área de memória não volátil, chamada de tabela *neighbor*. Na inicialização, se um dispositivo *child* conclui através da sua tabela *neighbor* que já fez parte de uma rede, ele pode executar um procedimento de notificação chamado *orphan*, para localizar a rede em que estava associado. Dispositivos que recebem a notificação *orphan* checarão suas tabelas *neighbor* e irão verificar se o dispositivo é seu *child*. Se verdadeiro, o dispositivo *parent* informará o *child* do seu lugar na rede. Se a notificação *orphan* falha ou o dispositivo *child* não possui nenhum *parent* em sua tabela *neighbor*, então ele tentará associar-se a rede como um novo dispositivo. Fazendo parte da rede, um dispositivo pode dissociar-se livremente dela ou ser dissociado pela requisição do seu dispositivo *parent*.

O montante de tempo que um dispositivo gasta determinando a energia dos canais e redes disponíveis em cada canal é especificada pelo parâmetro *ScanDuration*. Para a banda de 2,4GHz o tempo de exploração em segundos é calculado pela equação:

$$0,01536 * (2^{\text{ScanDuration}} + 1)$$

Se *ScanDuration* é ajustado para 8 e todos os 16 canais são definidos, o dispositivo gastará mais de 1 minuto executando a exploração de cada canal especificado. Roteadores e

dispositivos finais executam uma exploração para determinar as redes disponíveis, mas coordenadores executam duas, uma para amostrar a energia dos canais e outra para determinar a existência de redes. O tempo de exploração necessário precisa ser estimado, para adequadamente ser executado em cada canal, no montante de tempo destinado a inicialização da rede.

2 PROCEDIMENTO EXPERIMENTAL

2.1 Visão geral do projeto

O projeto proposto consiste de duas placas, que através de um link de RF, utilizando protocolo de rede ZigBee, trocam informações de captura de dados. No Anexo 2 é apresentado o diagrama elétrico do projeto e seu respectivo diagrama de blocos para melhor entendimento. Para efeito de redução de custos no projeto, tanto o circuito de captura / transmissão e o circuito de recepção, utilizam a mesma placa de circuito impresso. A placa de captura / transmissão é composta por todos os blocos do diagrama do Anexo 2, enquanto a placa de recepção é composta pelos blocos *FONTES*, *CPU*, *MUX / DEMUX*, *MÓDULO ZIGBEE*, *INTERFACE ICSP / ICE*, *INTERFACE SPI*, *I2C*, *DIGITAL / ANALÓGICA*, *INTERFACE USB* e *INTERFACE RS-232*.

A placa de circuito impresso foi projetada em quatro *layers*. Os dois *layers* externos são compostos por todas as *nets* do circuito, com exceção das de alimentação, que formam um dos *layers* internos da placa. Formam o *layer* de alimentação as *nets* *V_REG*, *+5VD*, *+5VA* e *+3,3VD*. Por fim o *layer* de GND, que comporta tanto o GND analógico como o digital. O *layout* deste *layer* foi realizado utilizando topologia de ligação estrela, portanto, o GND analógico e digital são conectados juntos próximos à fonte. Assim, nenhuma corrente provinda de componentes digitais passa através do GND analógico, como também, correntes provindas de componentes analógicos não atravessam o GND digital. Desta

maneira espera-se que níveis de ruído no GND analógico diminuam significativamente, melhorando a relação SNR e provendo uma melhor qualidade de conversão analógico digital.

São disponibilizados na placa de captura até oito canais para captura de dados. São utilizados na placa dois ADCs de 16 bits com interface unipolar (0 a 5V). Cada um possui quatro canais de entrada, e são amostradas duas entradas simultâneas por vez. Desta forma, há possibilidade de amostrar até 4 canais simultâneos por vez, característica desejada em algumas aplicações.

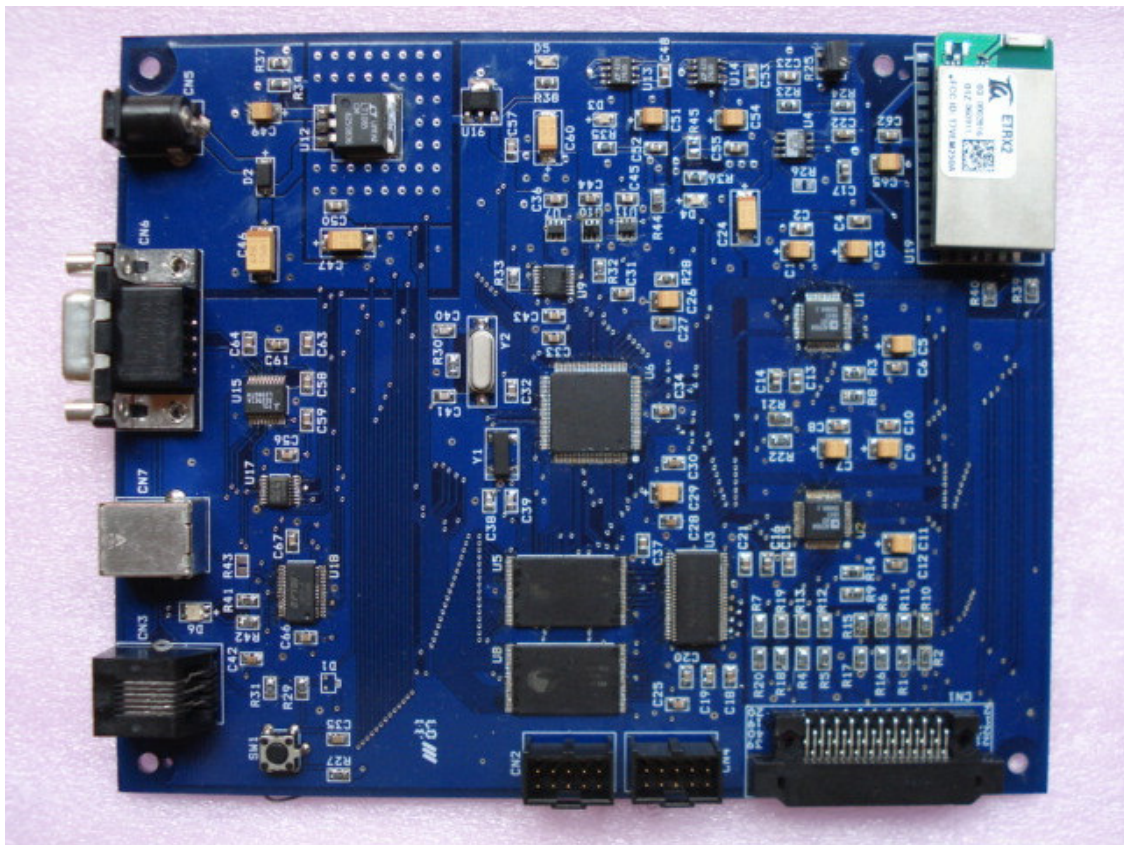


FIGURA 11: SISTEMA IMPLEMENTADO

As taxas de aquisição de cada canal podem ser configuradas individualmente. Contudo, existe uma dependência entre as taxas de amostragem entre os canais 0 e 4, 1 e 5, 2 e 6, 3 e 7. As taxas de amostragem entre estes pares de canais devem ser múltiplos entre

si. Por exemplo, se a taxa de amostragem do canal 0 é configurada para 10 ksps, a taxa de amostragem do canal 4 pode ser configurada para 10 ksps, 5 ksps, 3,33 ksps e assim por diante. As configurações da placa de aquisição podem ser passadas através de terminal remoto. Podem ser configurados quantos canais de aquisição são usados, taxas de aquisição, tamanho do *buffer* de armazenamento, entre outros.

O sistema, neste primeiro momento, realiza capturas sucessivas dos canais dos ADCs selecionados, até que o *buffer* de armazenamento esteja completo ou o ensaio seja interrompido pelo usuário. Após, os dados são transmitidos ao terminal remoto via módulos de comunicação que implementam o protocolo ZigBee. Com este esquema de funcionamento, taxas de aquisição maiores podem ser implementadas, pois os dados são transmitidos do *buffer* ao módulo de comunicação a uma taxa mais lenta.

2.2 Descrição do Hardware

A pesquisa e seleção de componentes adequados para a realização do projeto são fundamentais. Nos itens apresentados nesta seção são discutidas as principais características e, justificadas a escolha dos principais componentes dos circuitos implementados.

2.2.1 Fontes

A alimentação da placa é realizada por quatro reguladores de tensão lineares, cada qual alimentando blocos específicos de circuitos. A entrada de alimentação é positiva e possui um diodo em série, que protege a fonte contra inversões de polaridade que podem ocorrer na entrada. Para possibilitar que a tensão de alimentação da placa seja tão baixa quanto possível, sem a necessidade de circuitos elevadores de tensão, escolheu-se um diodo do tipo *Schottky* e um regulador de tensão *low dropout*. O diodo modelo B340LA (fabricado pela *Diodes Incorporated*) tem queda de tensão típica de 0,31V para correntes de 1A e suporta tensão reversa de ruptura de 40V.

O primeiro regulador de tensão, LT1085CM (*Linear Technology*), tem por função realizar uma pré-regulagem na tensão de entrada, para então ser fornecida aos outros reguladores do circuito. A queda de tensão típica entre seus terminais de entrada e saída, é de 1,3V para 1,5A. A tensão de saída foi projetada para aproximadamente 5,8V através da seguinte equação obtida no *datasheet* do fabricante:

$$V_{out} = V_{ref} \cdot \left(1 + \frac{R2}{R1}\right) + I_{adj} \cdot R2$$

$$V_{out} = 1,25 \cdot \left(1 + \frac{470}{130}\right) + 50\mu \cdot 470 = 5,79V$$

A diferença de tensão entre a entrada e saída do regulador não pode ultrapassar o valor de 30V. Assim, espera-se que a tensão de alimentação da placa possa variar de aproximadamente 7,65V à 36V, considerando a queda de tensão no diodo de entrada. A maior parte da dissipação de potência é, portanto, realizada por este circuito de pré-regulação, que está soldado numa grande área de cobre da placa, para auxiliar na dissipação de potência. Com esta topologia, os demais reguladores, alimentados pelo circuito pré-regulador, trabalham com tensões de entrada mais próximas das suas respectivas tensões de saída, dissipando menor potência, o que contribui para a estabilidade térmica dos circuitos e conseqüente redução de ruído, redução do componente e menor ocupação de área da placa de circuito impresso.

Os três reguladores que são alimentados pelo pré-regulador (LT1085CM), são os que fornecem efetivamente as tensões que são utilizadas pelos circuitos. Um regulador de 3,3V, com baixo ruído e rápida resposta transiente, modelo LT1963A, foi escolhido para alimentar todos os circuitos lógicos. Dentre os circuitos alimentados estão o bloco do microprocessador, módulo *ZigBee*, interfaces de comunicação (RS-232, USB), memórias e portas lógicas. O transceiver USB e os ADCs são efetivamente alimentados por fontes de 5V específicas, mas também utilizam a fonte de 3,3V para converter seus níveis lógicos para os mesmos níveis do restante do circuito.

Enfim, dois reguladores de 5V, da série LT1763, com baixo ruído (20μV_{RMS}), são

utilizados para alimentar o transceiver USB e os ADCs. Um dos reguladores é utilizado somente pelos circuitos analógicos dos ADCs, evitando-se assim, que o consumo de corrente impulsiva de circuitos lógicos produzam ruídos nesta linha de alimentação. As linhas de tensão digital dos ADCs podem ser alimentadas por qualquer um dos dois reguladores, através da montagem de um resistor de 27Ω (recomendado). Este resistor, em conjunto com o capacitor de desacoplamento, serve com um filtro RC, caso a fonte analógica alimente o circuito digital do ADC.

Em grande parte dos circuitos, seguindo as recomendações dos fabricantes, utilizaram-se capacitores de tântalo tanto para filtro, junto às saídas de cada regulador, como para desacoplamento. Capacitores de tântalo têm, em geral, baixa impedância e rápida resposta transiente, o que contribui significativamente para a redução de ruídos na placa.

2.2.2 Memória SRAM

Sistemas de aquisição geram grandes quantidades de informação e esta deve ser armazenada em algum meio. Estando a placa de aquisição não conectada a um computador, onde há uma vasta quantidade de memória que pode ser utilizada, uma solução é armazenar os dados em um buffer de memória *SRAM*. Outros meios, como memórias *DRAM*, que além de mais baratas são mais densas e rápidas, possuem sistemas de controle mais complexos, necessitando por vezes circuitos dedicados, para realizar o controle de leitura escrita e produção de ciclos de *refresh*.

Cartões de memória *FLASH* são densos e possuem custo / benefício bastante atrativos, mas necessitam de desenvolvimento de *software* razoável para serem utilizados adequadamente. Possuem tempo de latência elevado se comparado com memórias *SRAM* e *DRAM*, o que pode limitar a taxa de aquisição do sistema. Circuitos integrados de memória *E2PROM* e *FLASH* possuem limitações similares aos cartões de memória. Estes dois últimos tipos de memória e os cartões, tem vida útil limitada em alguns milhares de ciclos de escrita, enquanto memórias *SRAM* e *DRAM* tem ciclagem não determinada.

Por necessitar de um circuito de controle bastante simples, rápida, tipicamente ter baixo consumo de energia e ter ciclagem indeterminada, optou-se pela utilização de um buffer de memória *SRAM*. A memória selecionada foi a CY62167DV30, fabricada pela *Cypress*. Tem como principais características: tempo de acesso de 55ns, tensão de alimentação na faixa de 2,2V a 3,6V, consumo de corrente de 4mA a 1MHz, capacidade de 16Mbit, interface de dados 8 *bits* ou 16 *bits*. São utilizados duas pastilhas de memória, disponibilizando, portanto, um buffer de 4M *byte* para o armazenamento de dados. O barramento de endereços não é multiplexado, sendo necessários 20 linhas de endereço para acessar toda a memória. O controle de escrita / leitura é realizado pelas vias de *Output Enable* e *Write*. As duas vias de *Chip Select*, sendo uma delas negada, o que facilita a expansão de memória, controlam o acionamento da memória. São disponibilizados também três pinos de controle para acesso a *byte* (*BHE*, *BHL* e *BYTE*) e mais uma via de endereço que é multiplexada com o pino de dados *I/O15*, que não serão detalhados, pois a topologia projetada acessa a memória em barramento de 16 *bits*.

2.2.3 Conversores Analógicos / Digitais

Há uma gama enorme de ADCs para escolha. Algumas premissas foram impostas durante a decisão de qual ADC escolher: resolução de 16 bits, entrada unipolar (simplificando a implementação das fontes), o sistema deve ter ao menos 6 canais (preferencialmente simultâneas) e interface de dados paralela (diminui tempos de acesso à informação). Partindo destas premissas iniciais, optou-se pelo AD7654, fabricado pela *Analog Devices*.

O AD7654 possui interface paralela de 16 *bits* ou 8 *bits*, interface serial SPI[®], QSPI[™] e MICROWIRE[™]. Pode ser alimentado com até três fontes distintas, sendo uma de 5V nominal para seus circuitos analógicos, uma de 5V nominal para seus circuitos digitais e um terceira, na faixa de 2,7V a 5,25V, destinada a alimentar seus circuitos lógicos que realizam a interface externa (barramento de dados, linhas de controle, monitoração e

habilitação). Desta maneira, é possível interfacear o ADC com circuitos lógicos que operam em tensões nominais como 2,7V, 3V, 3,3V ou 5V.

São utilizados dois ADCs no circuito. Cada um possui quatro canais de entrada, sendo dois amostrados simultaneamente por vez. Dois circuitos *sample and hold* internos disponibilizam a tensão para ser convertida para código digital, por um circuito de aproximação sucessiva (SAR). A seleção de quais canais serão amostrados é realizado por uma linha de controle, designada A0, que realiza a multiplexagem dos canais para os circuitos de *sample and hold*. A conversão é iniciada assim que o pino *CVST* é colocado em nível lógico 0. O ADC possui dois modos de operação, controlados pela linha *impulse*. No modo normal, o consumo de energia é fixo, independente da taxa de amostragem, que pode chegar até 500ksps. No modo *impulse*, o consumo de energia é proporcional à taxa de amostragem, que pode chegar a 444ksps. Opcionalmente o ADC pode ser desabilitado, através do pino *PowerDown*, reduzindo seu consumo ao mínimo.

O término das aquisições podem ser avaliadas através de dois pinos diferentes. O pino *Busy*, indica quando ambos os canais já foram convertidos e, o pino *EOC* indica o término individual da conversão de cada canal. Optou-se pela utilização do pino de *Busy* de simplificação. A leitura dos canais convertidos é multiplexada no barramento de dados. O controle de qual canal será disponibilizado para leitura é realizado pelo pino denominado *A/B*.

O ADC7654 necessita de uma tensão de referência externa, igual à metade da tensão de fundo de escala das tensões de entrada analógica, que é de 5V. É recomendado, pelo próprio fabricante, o circuito integrado AD780BR. Este circuito integrado, provê uma referência de tensão de 2,5V de alta precisão e baixíssimo ruído, podendo ser menor que 20 μ V, pela utilização de capacitores de filtro, no valor de 100 μ F. Tem coeficiente de temperatura máximo de 3ppm/ $^{\circ}$ C, o que torna as conversões do ADC mais estáveis em uma faixa maior de temperatura.

2.2.4 Interfaces de Comunicação USB e RS-232

Hoje, a quase totalidade de computadores utiliza a interface USB como meio de comunicação com dispositivos externos e, inúmeros equipamentos de laboratório vêm adotando este padrão. A interface RS-232 ainda permanece como um dos principais meios de comunicação para periféricos, mas vem aos poucos, sendo substituída. Se tratando de microcontroladores, interfaces do tipo UART ainda são as mais utilizadas, enquanto a USB esta sendo integrada nos modelos mais recentes.

São disponibilizadas ambas as interfaces, RS-232 e USB, no projeto. O *transceiver* RS-232 selecionado foi o ICL3225E, produzido pela *Intersil*. Possui duas linhas de transmissão e duas de recepção, possibilitando a implementação das linhas de controle de dados RTS e CTS, as mais comumente utilizadas com microcontroladores. Alguns pinos adicionais de controle possibilitam: identificar se o *transceiver* está conectado a interface RS-232 (*Invalid*), identificar se está pronto para iniciar comunicação (*Ready*), forçar o componente a entrar em operação (*ForceOn*) e forçar componente a desligar-se (*ForceOff*). Este *transceiver* suporta taxa de dados de 1Mbps e opera em tensões na faixa de 3V a 5,5V.

Para simplificar a implementação da interface USB, foi selecionado o *transceiver* FT232RL, produzido pela FTDI. Trata-se de um conversor de protocolo USB para UART. Opera na especificação USB 1.1 / USB 2.0 *full speed* (12Mbps) do protocolo USB e, a interface UART opera entre as taxas de 300bps e 3Mbps. Possui buffer, tipo *fifo*, de 128 *bytes* para transmissão e 256 *bytes* para recepção. O componente opera entre as tensões de 3,3V a 5,25V e, possui entrada adicional de tensão utilizada para conversão dos níveis lógicos para valores inferiores ao da alimentação nominal (1,8V a Vcc).

2.2.5 Módulo ZigBee

Projetos de RF tendem a consumir horas de desenvolvimento e requerem extenso

know-how por parte dos projetistas e, mesmo assim, nem sempre os resultados são os esperados. O desenvolvimento e adaptação do software do protocolo ZigBee também gera extensas horas de trabalho e testes. Portanto, para minimizar o tempo de desenvolvimento e evitar séries de problemas, optou-se por uma solução comercial. Há disponível no mercado uma grande variedade de módulos de comunicação que estão sendo homologados no protocolo ZigBee. O módulo selecionado para o projeto foi o ETRX2, fabricado pela *Telegesis*.

O módulo tem duas opções de conexão, um conector, não montado, de 20 vias ou integração direta na placa de aplicação através de 38 *pads* para soldagem SMD. Possui 12 pinos de I/O, dois ADCs de 10 *bits*, um pino exclusivo para *reset*, uma UART (RX, TX, CTS, RTS) e interface para programação e depuração do microcontrolador integrado. Alguns dos pinos de I/Os têm funções especiais como interrupção e PWM, mas não são usados na aplicação final do projeto. Utiliza a banda de frequência de 2,4GHz para comunicação, podendo ser configurado em qualquer um dos 16 canais definidos pelo protocolo ZigBee. Suporta a faixa de tensão de 2,1V à 3,6V. Tem consumo típico de 41,5mA, para potência de transmissão de +5dBm e, consumo de 37,5mA, para sensibilidade de recepção típica de -97dBm. Pode atuar como coordenador, roteador ou dispositivo final. Opera somente com a topologia de rede *mesh*.

A conversação com o módulo é realizada através de comandos AT próprios, similares ao padrão *Hayes*, enviados à interface UART, que pode ser configurada para taxas específicas, que variam de 1200bps à 115200bps. A interface UART tem inúmeras outras opções de configuração, dentre elas: controle por fluxo de *hardware*, utilização de 1 ou 2 *bits* de parada, comunicação com 7 ou 8 bits, paridade par ou ímpar, eco, etc.

Cinquenta e dois registradores, todos acessados por comandos AT, são responsáveis pelo funcionamento do módulo em questão. A maior parte destes registradores podem ser acessadas tanto localmente, quanto remotamente. Com poucas exceções, os registradores do módulo são não-voláteis. Quatro registradores, por armazenarem as mais importantes informações de configuração, são protegidos por senha (*Encryption Key*, *OEM Word*, *Main Function*, *Password*). Pode-se dividir os registradores em cinco blocos principais: 4

registradores para a configuração do link de rádio (S00 – S03), 7 registradores para configurações gerais do módulo (S04 – S0A), 10 registradores para a configuração das I/Os (S0B – S14), 25 registradores relacionados a funções de software integradas ao módulo (S15 – S2D) e 6 registradores específicos do ETRX2, que diferem de outros módulos fabricados pela mesma empresa (S2E – S33). Os principais registradores usados durante o desenvolvimento do projeto serão abordados em capítulo posterior.

Todos os pinos de I/O podem ser acessados por comandos AT específicos, tanto localmente, quanto remotamente. Dos registradores que realizam o controle das I/Os, alguns são duplicados, sendo um volátil e outro não-volátil. Assim, se durante a operação do módulo, por exemplo, uma das portas de I/O foi mudada de entrada para saída, num novo ciclo de *start-up*, a porta volta a sua funcionalidade previamente configurada no registrador não-volátil.

Estão integradas à funcionalidade do módulo, 50 funções de software, que podem ser disparadas por dois pinos de I/O, que tem função de interrupção, assim como por qualquer um dos 8 timers disponíveis na arquitetura interna do módulo. Com a configuração adequada dos registradores e habilitação de algumas das funções integradas, dependendo da aplicação desejada, não é necessário a integração do módulo com um microcontrolador de suporte, até porque, a quase totalidade de registradores pode ser acessada remotamente. Upgrades do firmware do módulo também podem ser realizados remotamente.

2.2.6 Microcontrolador

Inúmeros fabricantes e modelos de microcontroladores estão disponíveis, mas também se deve levar em conta as ferramentas necessárias para o desenvolvimento do projeto de software. Devido a facilidade e custo das ferramentas, foram selecionados microcontroladores do fabricante *Microchip*. O ambiente de programação é disponibilizado na própria página do fabricante para download, assim como o compilador C, versão para estudante, que são livres de taxas de aquisição e não possuem período de avaliação

limitado. Para gravação e depuração, foi adquirida a ferramenta ICD2BR, fabricado pela LabTools, sob licença da *Microchip*.

Para realizar a integração e controle de todos os blocos, citados nos capítulos anteriores, é necessário um microcontrolador com grande capacidade de I/O. Os ADCs e as memórias SRAM selecionadas são de 16 *bits* e, a escolha de um microcontrolador de arquitetura de 16 *bits* é natural. Foi previsto que o *software* a ser desenvolvido, utilizaria uma quantidade de memória razoável, devido ao módulo ZigBee, que utiliza *strings* para comunicação e controle. *Softwares* que fazem utilização constante de *strings* costumam ocupar uma grande quantidade de memória de programa e também memória de execução, devido ao uso constante de funções de entrada / saída (exemplo: *printf* e *scanf*). A velocidade de execução do programa também é importante, para que taxas de aquisição na ordem de milhares de amostras por segundo seja alcançada. Neste ponto, decidiu-se pelo uso do microcontrolador PIC24FJ128GA010.

O PIC24FJ128GA010 possui 128k *bytes* de memória de programa e 8k *bytes* de memória de dados, que devem ser suficientes para satisfazer as necessidades do projeto. Disponibilidade de até 84 I/Os. Inúmeras interfaces para comunicação serial, sendo: duas UARTs (RS-232, RS-485, IrDA[®]), duas interfaces SPI (3-wire / 4-wire) e duas interfaces I²C[™]. Dezesesseis conversores ADCs com 10 *bits* de resolução. Cinco *timers* de 16 *bits*, podendo ser configurados como *timers* de 32 *bits*. Tem capacidade para execução de 16 MIPS (milhões de instruções por segundo) operando à 32MHz. Disponibilidade de um RTCC implementado em *hardware*. Estas são algumas das características do microcontrolador selecionado.

2.2.7 Placa de Circuito Impresso

Uma placa de circuito impresso foi projetada para integrar os componentes do projeto. Devido a quantidade de componentes, sendo a maior parte SMD e com espaçamento entre terminais de 0,5mm, o *layout* da placa tornou-se bastante denso,

necessitando de quatro camadas para seu roteamento. A documentação da placa é disponibilizada para consulta no Anexo 3.

Nas camadas de FC e FS foram dispostas todas as trilhas do circuito, com exceção das trilhas de GND, +5VD, +5VA, +3,3VD, que são roteadas, também, em duas camadas internas. Esta disposição permite, pelo acesso a todas as trilhas, que algumas alterações sejam possíveis, com a placa já pronta, caso identifique-se algum problema de projeto.

Na camada de “plano de alimentação”, estão presentes todas as trilhas que fornecem alimentação aos componentes da placa. São em geral trilhas mais robustas que as das camadas de FC e FS, para que a queda de tensão seja mínima quando há consumo elevado de corrente. Pelo aumento da área de cobre nessas trilhas, evita-se também a formação de ruídos modulados pelo consumo de corrente / queda de tensão nas linhas de alimentação.

O “plano de terra” da placa é dividido em duas áreas distintas: o plano de terra digital e o plano de terra analógico. O *layout* da placa foi concebido de forma que circuitos digitais e analógicos são dispostos e confinados em locais distintos, permitindo que os planos de terra analógico e digital ficassem “isolados” um do outro. Na parte inferior da placa são dispostos todos os circuitos analógicos: ADCs, fonte de tensão e referência de tensão. Estes circuitos estão referenciados ao plano de terra analógico. Neste plano não circulam “correntes digitais” (com exceção das próprias “correntes digitais” da interface digital do ADC) tornando ele uma área “silenciosa” para que as conversões AD possam ocorrer com o mínimo de ruído possível [5]. A união dos planos de terra digital e analógico ocorrem próximo ao pré-regulador de tensão, seguindo uma topologia estrela. Espaços vazios nas camadas de FC e FS também foram usados para compor o plano de terra, seguindo os mesmos cuidados de isolamento.

2.2.8 Topologia de Hardware Implementada

No Anexo 2, são apresentados o diagrama em blocos e o esquema elétrico do projeto, que devem ser consultados para melhor compreensão, no decorrer do

desenvolvimento deste item.

O barramento de dados de 16 *bits* do sistema, é realizado pela “porta D” do microcontrolador. As duas memórias SRAM e os ADCs são conectados a este barramento. Para minimização de ruídos nos circuitos analógicos da placa, o barramento de dados dos ADCs, são isolados do barramento de dados das memórias e microcontrolador, através de uma chave de barramento. Transição de sinais digitais, no barramento de dados dos ADCs, podem introduzir ruído durante a operação de conversão. O circuito integrado SN74CB3Q1644, de fabricação da *Texas Instruments*, é uma chave de barramento bi-direcional de 16 bits. Quando desabilitada, funciona como um circuito de alta impedância entre os barramentos a serem isolados. Quando habilitada, fornece acesso bi-direcional, com impedância típica de 5Ω , sem os inconvenientes pinos de controle para seleção de direção do fluxo de dados.

Para evitar o acionamento acidental de dois barramentos de dados ao mesmo tempo, foi implementada uma lógica de controle auxiliar. Este circuito é composto pelo integrado SN74LVC139A, que são dois decodificadores de 2 para 4 linhas e, três portas lógicas configuráveis, SN74AUP1G58, sendo duas operando como portas do tipo NAND e uma operando como XOR. O circuito é controlado por quatro pinos do microcontrolador e, tem por função habilitar somente o barramento de dados do circuito integrado que deseja-se acessar (SRAM#0, SRAM#1, ADC#0, ADC#1). Sempre que um dos ADCs for acessado, a lógica de controle habilita também a chave de barramento, para que a comunicação com o microcontrolador seja possível. Na tabela 3 é apresentada a tabela verdade do circuito implementado.

TABELA 3 – TABELA VERDADE PARA ACIONAMENTO DOS BARRAMENTOS DE DADOS

| Linhas de controle | | | | ADC#0 | | ADC#1 | | SRAM | |
|--------------------|----------|----------|----------|-------|------|-------|------|------|------|
| CTR_PER3 | CTR_PER2 | CTR_PER1 | CTR_PER0 | CS0# | RD0# | CS1# | RD1# | CS2# | RD2# |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | x | X | 1 | 1 | 1 | 1 | 1 | 1 |

Os 21 *bits* necessários, para gerar o barramento de endereços, foram implementados utilizando-se as portas: A (de RA0 à RA7), E (de RE0 à RE7) e G (RG0, RG1, RG12 a RG14). O microcontrolador escolhido, não possui uma interface direta para acesso a uma memória tão extensa. Uma solução em *software* é implementada para gerar corretamente os endereços de acesso à memória SRAM.

A interface de controle dos ADCs, com exceção dos pinos de seleção (CS) e leitura (RD), é diretamente conectada ao microcontrolador. Cada ADC pode ser controlado individualmente (seleção de canal para conversão, seleção de canal para leitura e solicitação de conversão), com exceção do modo de operação, que é configurado pelos pinos *IMPULSE* e *PD*, comuns aos ADCs. O término das conversões é indicado quando o pino *BUSY* apresenta nível lógico zero.

Um conector de 40 vias faz a interface externa com os canais dos ADCs. Todos os canais analógicos dos ADCs e seus respectivos GNDs estão disponíveis neste conector. Para facilitar a implementação de circuitos condicionadores, as fontes de 5V (digital e analógica), 3,3V e o GND digital da placa são também disponibilizados para uso nesta interface. O chicote confeccionado para utilização, é constituído de dois cabos tipo *flat* de 20 vias cada. A distribuição dos sinais neste conector, foi realizada de forma que as vias que

conduzem os sinais analógicos, estão entre duas vias de GND (analógico), que servem de blindagem para os sinais conduzidos pelo cabo.

Três são os dispositivos que necessitam de comunicação serial: o *transceiver* RS-232, o *transceiver* USB e o módulo ZigBee. O microcontrolador em questão possui somente dois módulos de comunicação UART, sendo necessário mais um circuito lógico para resolver esta limitação. O circuito integrado SN74CB3Q3257, fabricado pela *Texas Instruments*, é um multiplexador / demultiplexador de 1 para 2 de 4 *bits*, com fluxo de dados bi-direcional. Com a implementação deste circuito, as interfaces USB e RS-232 podem ser multiplexadas numa única UART. A seleção de qual interface será utilizada para comunicação, pode ser realizada com auxílio dos próprios *transceivers*, que possuem pinos que indicam se existem níveis de tensão válidos nos seus terminais de entrada, indicando, portanto, se ele está conectado a um terminal de comunicação ativo. No *transceiver* RS-232 esta indicação é realizada pelo pino *invalid*, que apresenta nível lógico 1 quando o componente esta conectado a uma porta RS-232. Similarmente, o *transceiver* USB possui um pino com a mesma funcionalidade, denominado CBUS4. Ambos os pinos são conectados ao microcontrolador que decide, pelo estado dos níveis lógicos indicados, se a interface USB ou RS-232 será selecionada para comunicar com sua interface UART.

A conexão com a interface USB é realizada com cabos do tipo A-B. O conector tipo A é o padrão para computadores (*hosts*) e o conector tipo B utilizado como padrão para conexão de dispositivos (*scanners* e impressoras por exemplo) . A interface RS-232 foi projetada de forma que a placa seja tratada como um dispositivo do tipo DCE (*Data Carrier Equipment*), sendo necessário a confecção de cabos tipo *straight thru* com terminações macho / fêmea. É disponibilizado na interface RS-232 as linhas de RTS e CTS para controle do fluxo de dados.

O módulo ZigBee ocupa a UART restante do microcontrolador. Utiliza somente as linhas de RX e TX para comunicação. O pino de *reset* do módulo é conectado ao microcontrolador, para que o módulo possa ser reiniciado, caso ocorra algum problema durante sua operação. O pino de I/O0 do módulo, que possui função de interrupção, é disponibilizado ao microcontrolador, para implementação de alguma funcionalidade não

prevista durante o desenvolvimento do projeto.

2.3 Descrição do Software

Os softwares descritos nos itens 2.3.2 e 2.3.3 foram escritos utilizando o ambiente de programação MPLAB versão 7.5 e compilados com o compilador C30 2.05, ambos disponibilizados pela Microchip.

2.3.1 Configurações realizadas nos Módulos ZigBee

Antes da integração dos módulos de comunicação ZigBee, foram realizadas algumas configurações nos seus registradores, para que os mesmos pudessem atender as necessidades dos softwares implementados nas placas de aquisição e recepção. Segue um descritivo dos registradores que tem configuração comum à ambos os módulos e suas respectivas funcionalidades.

S00 - Máscara de canais: este registrador não foi alterado. Ele representa os 16 canais onde uma rede pode ser estabelecida. É padrão de fábrica todos os canais estarem habilitados. Quando o módulo ZigBee inicia uma rede, ele realiza uma leitura de energia de todos os canais e estabelece sua rede no canal que possuir menos energia. Portanto, deixando todos os canais habilitados, a rede sempre será estabelecida nos canais que têm o menor tráfego de dados, o que favorece menor erro nas transmissões / recepções.

S01 - Identificação preferida para a PAN: este registrador não foi alterado. A identificação da rede pode variar de 0x0000 à 0x3FFF. Se um número nesta faixa estiver configurado no registrador, a rede será estabelecida com este número de identificação e, somente módulos com a mesma identificação poderão associar-se a rede. Se uma rede já possuir este número de identificação, no mesmo canal, o módulo gera automaticamente um número aleatório para estabelecer sua rede. Caso o número no registrador for configurado

para 0xFFFF, a rede será estabelecida com um número de identificação aleatório. Os módulos vêm configurados de fábrica com o valor 0xFFFF.

S02 - Potência de transmissão: para aumentar a faixa de atuação do módulo, configurou-se este registrador com o valor “4”, que segundo o datasheet aumenta a potência de transmissão para +5dBm e a sensibilidade para -97dBm.

S03 - Chave de encriptação: é padrão que todos os módulos tenham este valor igual a zero, assim, qualquer módulo pode associar-se a qualquer rede. Para ter maior controle e evitar que módulos que não fazem parte da aplicação desenvolvida associem-se a esta rede específica, consumindo a banda de dados, uma chave de encriptação foi fornecida aos módulos da placa de transmissão e placa de recepção.

S09 – Senha: registradores que armazenam informações importantes de configuração são protegidos por senha, assim como alguns comandos. Como registradores de configuração e comandos podem ser acessados e fornecidos remotamente, por outra aplicação, é interessante proteger a rede contra acessos indevidos.

S0B - Configurações da UART: apenas foi desabilitada a função de ECO (bit 4 igual a 1), para simplificar o software de controle. As outras características da UART, que não foram alteradas são: *baud rate* de 19200, 8 *bits* de comunicação, 1 *bit* de parada, sem paridade, sem controle de fluxo por *hardware*.

2.3.1.1. Configurações do Módulo ZigBee na Placa de Recepção

S04 – Nome do nó: nome que pode ser utilizado pelo usuário para identificar o nó na rede. O módulo pode ser identificado pelo EUID64, que é único, mas no caso de substituições, evita-se que o software de aplicação que controla o nó, tenha de ser atualizado. A placa de recepção é identificada no software de aplicação pela *string* “PCI_AQS”.

S06 – Função principal: a placa de recepção será a responsável pela inicialização da

rede, transmissão das informações de configuração para a placa de aquisição e recepção dos dados adquiridos nos ensaios. Por este motivo foi configurada como um dispositivo FFD. Neste registrador são disponíveis as configurações de outras funcionalidades importantes. Foram habilitadas também as opções: associação segura à rede utilizando a chave de encriptação e, espera de confirmação no caso de transmissões do tipo UCAST e SCASST. Para ativar estas opções o valor de carga do registrador S06 é de 0x2051.

S2E – Registrador específico do dispositivo: apresenta configurações que diferem de outros produtos que utilizam o mesmo firmware. Foram zerados os últimos dois *bits* deste registrador, para desabilitar a interrupção do pino de I/O0.

S0E – Valor inicial do registrador S0D: o registrador S0D é um registrador volátil que controla o a direção dos dados das portas de I/O do módulo. S0E é o registrador não-volátil, cujo valor será copiado para o registrador S0D toda vez que o módulo for iniciado. O *bit* zero deste registrador foi alterado para o valor 1, configurando a porta de I/O0 como saída.

Registradores de S17 à S1E: estes registradores controlam quais as funções e tempos em que serão executadas, cada vez que um dos 8 *timers*, disponíveis no microcontrolador integrado ao módulo, produz uma interrupção por *overflow*. Para o módulo da placa de recepção, todos as funções disparadas pelos *timers* foram desabilitadas.

2.3.1.2 Configurações do Módulo ZigBee na Placa de Aquisição

S04 – Nome do nó: a *string* de identificação da placa de aquisição é “PCI_AQS”.

S06 – Função principal: o módulo pode ficar longos períodos sem necessitar de comunicação contínua com a placa de recepção. Por este motivo o módulo integrado nesta placa foi configurado como um dispositivo RFD. O valor de carga neste registrador é 0x2251. Outras funções habilitadas pela carga deste valor são: associação segura a rede utilizando chave de encriptação e, espera por confirmação nas transmissões de pacotes do

tipo UCAST e SCAST.

Registradores S0E e S2E: idem a descrição apresentada no item 2.3.1.1.

Registradores S17 e S18: controlam, respectivamente, a função e o tempo de execução relacionados ao *timer* 0 do módulo ZigBee. Foi utilizada a função integrada número 0x0011. A função foi configurada para ser executada a cada segundo, passando o valor 0x0004 para o registrador S17 (cada interação no timer equivale a 250ms). A função 0x0011 é utilizada somente por RFDs (*mobile / sleepy*). RFDs configurados com esta função, realizam um pedido periódico (que depende do valor de estouro do *timer*) junto ao seu *parent*, para a transmissão de dados pendentes que são destinados à ele. Na aplicação proposta, isso significa que a cada 1 segundo, a placa de transmissão irá verificar se existem dados que são destinados à ela, junto a placa de recepção.

Registradores S19 a S1C: configurados com valor 0x0000. Funções dos *timers* desabilitadas.

Registradores S1D e S1E: controlam, respectivamente, a função e o tempo de execução relacionados ao *timer* 3 do módulo ZigBee. Foi utilizada a função integrada número 0x0016. O período de execução estipulado foi de 1 minuto, passando o valor 0x00F0 para o registrador S1E. Esta função verifica se o módulo faz parte de uma rede. Se o módulo não está associado a nenhuma rede, a função força que o módulo procure uma e associe-se a mesma. Na aplicação proposta, isto significa que a cada um minuto, a placa de aquisição irá procurar pela rede estabelecida pela placa de recepção.

2.3.2 Software da Placa de Recepção

O *software* desenvolvido para a placa de recepção é responsável pela interface com o usuário. Para tornar o sistema mais simples de ser utilizado, o usuário não precisa efetuar nenhuma configuração de rede para realizar a comunicação com a placa de aquisição. A entrada e saída de dados é feita com o uso do *software Hyper Terminal*. As configurações

necessárias para comunicação são: *baud rate* de 19200, 8 *bits* de comunicação, 1 *bit* de parada, sem paridade e sem controle de fluxo.

Para que a placa de recepção possa ser usada pela interface USB, é necessário a instalação do *driver* VCP versão 2.00.00, disponível no site da FTDI. Este *driver* faz com que o dispositivo USB conectado, apareça no sistema operacional, como se fosse uma porta COM.

2.3.2.1 Execução da Função Principal

A utilização do sistema é bastante simples. É apresentado na figura 12, 13 e 14, o fluxograma de execução do *software* principal implementado no microcontrolador. Com a placa devidamente alimentada, conecta-se ela ao PC, pela interface RS-232 ou USB. O *software* da placa de recepção detecta automaticamente a interface que esta ativa, não necessitando de nenhuma intervenção para configuração. Abre-se o *software Hyper Terminal* e realizam-se as configurações já mencionadas no item anterior. Assim, a placa esta pronta para entrar em funcionamento.

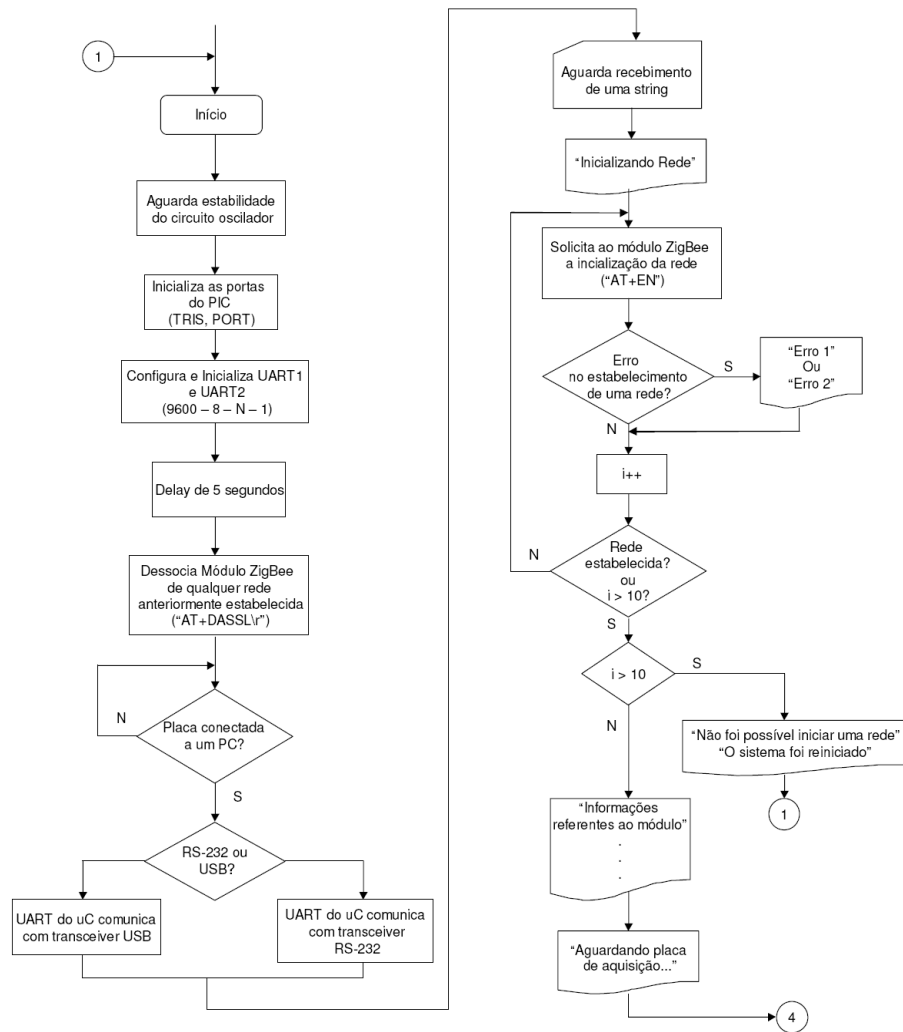


FIGURA 12: FLUXOGRAMA DO SOFTWARE PRINCIPAL DA PLACA DE RECEPÇÃO

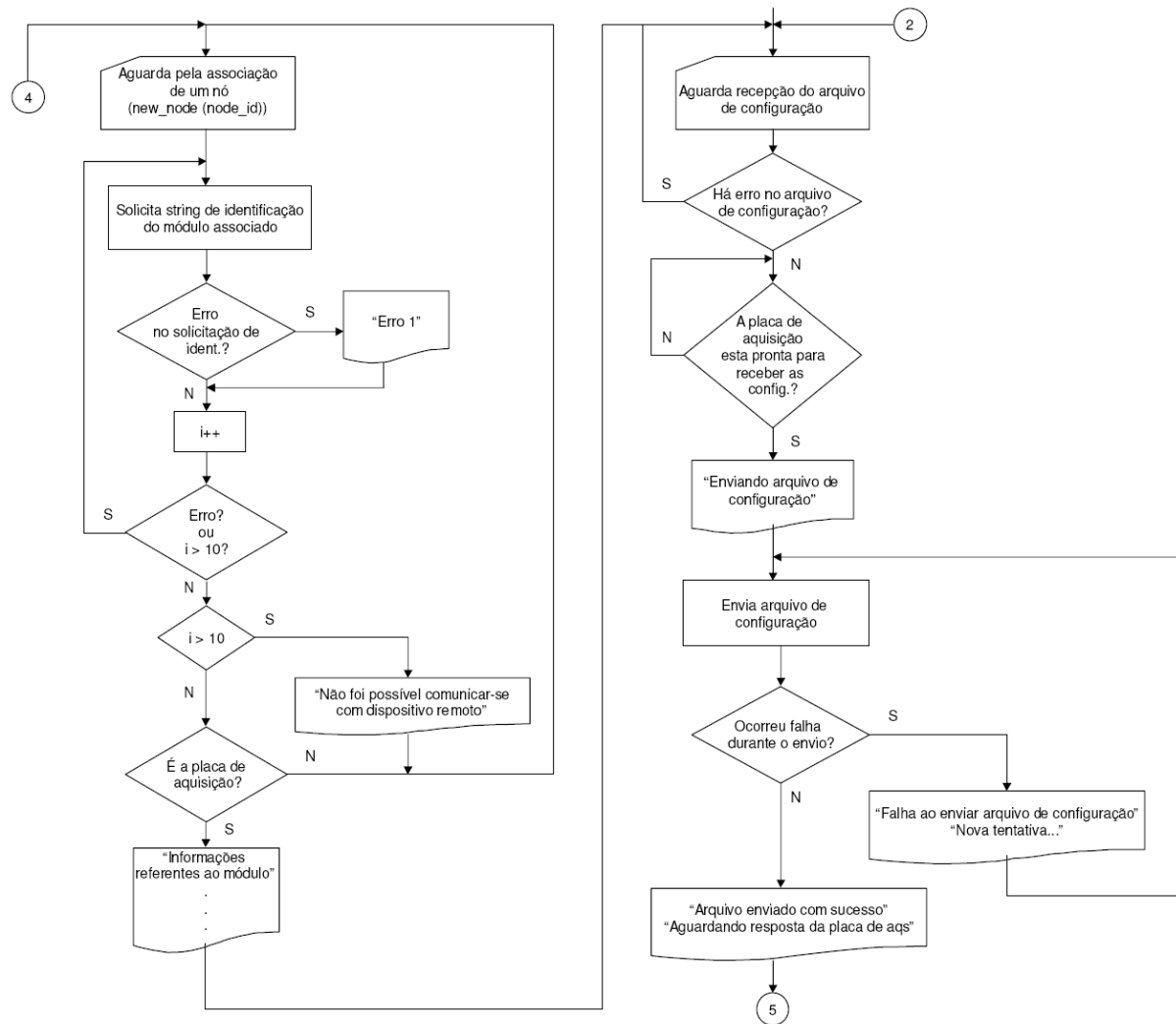


FIGURA 13: FLUXOGRAMA DO SOFTWARE PRINCIPAL DA PLACA DE RECEPÇÃO

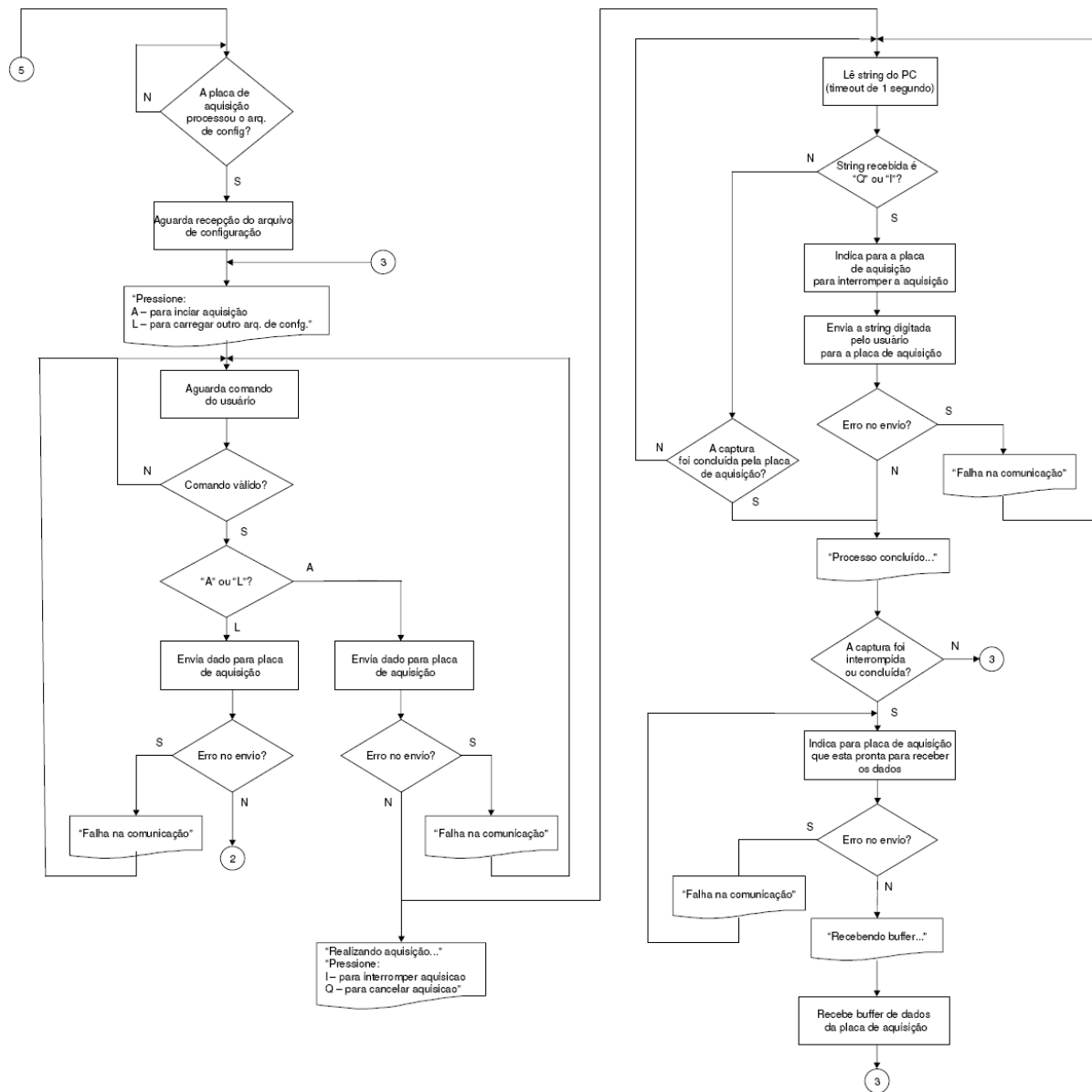


FIGURA 14: FLUXOGRAMA DO SOFTWARE PRINCIPAL DA PLACA DE RECEPÇÃO

O processo de inicialização de uma rede começa assim que qualquer *string* for digitada na tela do terminal. A mensagem “*Inicializando Rede*” é apresentada na tela. Este processo demora pouco menos de 20 segundos. Quando a rede é estabelecida com sucesso, são apresentados no terminal, a identificação EUID64 do módulo, o número de identificação da rede, canal onde ela foi estabelecida e o nome dado ao nó. Se a rede não

consegue ser estabelecida após 10 tentativas, uma mensagem de erro é apresentada ao usuário e o *software* é reiniciado.

Com a rede já estabelecida, o receptor fica indefinidamente esperando que a placa de aquisição associe-se à rede. Assim que um nó associa-se à rede, o *software* requisita a palavra de identificação do nó. Se a palavra de identificação for a da placa de aquisição (PCI_AQS), as informações do nó são apresentadas na tela (EUID64 e nome do nó), caso contrário a espera pela associação do transmissor continua.

Logo após o estabelecimento da comunicação com a placa de aquisição, é solicitado ao usuário, que entre com o arquivo de configuração contendo as taxas de amostragem de cada canal e o tamanho do buffer a ser utilizado. O arquivo tem o seguinte formato:

```
CH_0  1000
CH_1  1000
CH_2  1000
CH_3  1000
CH_4  1000
CH_5  1000
CH_6  1000
CH_7  1000
BUFFER      2000
```

As taxas de amostragem podem variar desde 1sps até 5ksps. O tamanho do *buffer* pode ser configurado até o valor de 2097152 *words*. Caso haja erro de sintaxe no arquivo, o mesmo é descartado e informado ao usuário para tentar novamente.

O arquivo é automaticamente transferido para a placa de aquisição. O usuário é informado se o arquivo foi recebido, ou se ocorreu algum erro durante a transmissão. Se ocorreram erros, nova tentativa é realizada dentro de alguns segundos. Assim que o arquivo é recebido com sucesso pela placa de aquisição, as mensagens “*Arquivo enviado com sucesso!*” e “*Aguardando resposta da placa de aquisição...*” são exibidas na tela. A placa de aquisição envia uma resposta quando termina de realizar todas as configurações informadas no arquivo transmitido.

Um menu com duas opções é exibido na tela do *Hyper Terminal* quando a placa de aquisição termina de efetuar suas configurações:

```
Pressione:
A – para iniciar aquisição
```

L – para carregar outro arquivo de configuração

Durante a fase de aquisição, o usuário tem a opção de descartar o ensaio em andamento, ou interrompê-lo e receber os dados adquiridos até o momento.

Pressione:

I – para interromper a aquisição

Q – para cancelar a aquisição

A transmissão dos dados, da placa de aquisição para a de recepção, é iniciada assim que não haja mais espaço nos *buffers* de memória ou a opção “*interromper*” seja digitada pelo usuário. Os dados são exibidos na tela numa formatação que segue: identificação do canal (CH_X), nova linha, tabulação, dado, nova linha ..., identificação de fim (END). Para melhor compreensão, segue exemplo.

```
CH_0
    10235
    23495
    04958
    ...
    49853
END
CH_1
    32987
    09384
    ...
    30945
END
...
CH_7
    34844
    ...
    56542
END
```

Ao final do recebimento dos dados, o *software* volta para o menu de aquisição, solicitando ao usuário para iniciar nova aquisição ou carregar novo arquivo de configuração.

2.3.2.2 Descrição das Funções Implementadas

O módulo de comunicação ZigBee comunica-se basicamente através de *strings*

enviadas através de sua interface UART. Na placa de recepção ele é o principal componente e, faz-se neste *software*, uso constante de funções para manipulação de *strings*. Grande parte são funções de bibliotecas, fornecidas junto com o compilador C30, outras tiveram de ser desenvolvidas devido ao uso muito específico, ou dificuldades de implementação das funções de biblioteca.

A função *int read_string_uart_zigbee(unsigned char *BUFFER, unsigned int LENGH, unsigned char TIMEOUT)*, foi desenvolvida para substituição da tradicional *scanf()*, que poderia ter sido usada como alternativa. Devido as respostas do módulo ZigBee seguirem o formato <CR><LF>*dados*<CR><LF>, a implementação da função *scanf()* não se torna muito adequada. Três são os argumentos passados para a função: o ponteiro onde a *string* será armazenada, o tamanho máximo permitido para a *string* e o tempo máximo que a função pode ficar sem receber um caractere. Assim que a função é chamada, a contagem de *timeout* é iniciada. Se não for recebido nenhum caractere durante o tempo especificado, a função retorna o código de erro -3. Caso um caractere seja recebido, o contador de *timeout* é reiniciado. Os caracteres iniciais de LF e CR são desprezados. O armazenamento da *string* inicia quando o primeiro caractere diferente de LF ou CR é recebido. Para cada caractere recebido, é verificado se o tamanho máximo da *string* não foi ultrapassado. Se o tamanho máximo da *string* for ultrapassado, é inserido um caractere nulo na primeira posição do ponteiro, para evitar que a *string* seja tratada no *software*. O código de erro -2 é informado nesta situação. O armazenamento dos caracteres continua até que uma LF ou CR seja recebido novamente, então, para finalizar a *string*, é inserido um caractere nulo na posição seguinte a do último caractere recebido. Quando a função é executada com sucesso, o código 0 é retornado por ela. A função *int read_string_uart_pc()* tem o mesmo funcionamento, mas é direcionada para a UART2 do microcontrolador.

A função *int init_WPAN(char *device_id, char *wpan_ch, char *wpan_id)* é responsável pela inicialização de uma rede. São passados três ponteiros para a função, que irão armazenar as informações da rede estabelecida. As informações são: identificação EUID64 do módulo integrado à placa, canal onde a rede será estabelecida e número de identificação da rede. Todos são dados formatados em *strings*. A identificação EUID64 do

módulo é requisitada com o comando “ATT”, que retorna também outras informações que são ignoradas. Para o estabelecimento de uma rede, o comando “AT+EN” (*establish PANetwork*) deve ser enviado ao módulo ZigBee. Após cerca de 16 segundos, são retornados o número de identificação da rede e canal onde foi estabelecida. Dois erros podem ser retornados pela função: falha na comunicação com o dispositivo ou falha no estabelecimento de uma rede.

A função *int new_node (char *node_id)* aguarda a associação de novos nós ao coordenador. Toda vez que um nó associa-se a rede, uma mensagem “NEWNODE:<EUI64> ” é recebida pelo *parent*. A identificação EUI64 é armazenada no ponteiro informado no argumento da função. O único código de erro retornado, -1, é uma indicação de *timeout*, pois a função permanece procurando por novos nós somente por dois minutos.

Para transmissão de dados foram escritas duas funções: *int unicast (char *node_id, char *string, char x)* e *int broadcast (char *hops, char *string, char x)*. A função *unicast* realiza transmissões endereçadas. São passados como argumentos para a função, o ponteiro que contém a identificação EUI64 do módulo para o qual deseja-se transmitir um pacote de dados, o ponteiro da *string* com os dados a serem enviados e o número de tentativas para transmissão do pacote. Passados os argumentos, a função monta o comando para envio do pacote, AT+UCAST:<EUI64>,<DATA>, e transmite esta *string* para o módulo integrado à placa. É aguardado então a confirmação do recebimento do pacote de dados pelo nó remoto. Se não há confirmação de recebimento, a função é executada tantas vezes quanto as especificadas pelo argumento “x”. Se após o número de tentativas máxima não houver a confirmação de recebimento pelo nó remoto, a função retorna com o código de erro -1.

A função *broadcast* realiza transmissões não endereçadas. São passadas como argumento a quantos nós sequenciais o pacote pode ser transmitido, o ponteiro da *string* que contém os dados a serem enviados e o número máximo de tentativas para transmissão do pacote. A função monta a *string* de comando, AT+BCAST:nn,<DATA>, onde nn representa o argumento **hops*, que pode variar de 00 a 10. A função envia esta *string* ao módulo integrado à placa e aguarda uma resposta de transmissão bem sucedida. São

realizadas mais “x” tentativas de transmissão caso não a transmissão falhe. Se o módulo não conseguir realizar a transmissão, o código de erro -1 é retornado pela função. Com as funções *unicast* e *broadcast* são possíveis a transmissão de até 65 *bytes* de dados.

Todos os pacotes de dados recebidos pelos módulos ZigBee através de transmissões do tipo *unicast* e *broadcast* tem, respectivamente, o formato UCAST:<EUI64>=<DATA> e BCAST:<EUI64>=<DATA>. Para facilitar a manipulação destas *strings*, foi escrita a função *int str_inf (char *string, char *node_id, char *str_inf)*. São passados como argumento para a função, respectivamente, o ponteiro da *string* a ser manipulada, o ponteiro para armazenamento da identificação do nó que enviou o pacote de dados e o ponteiro onde será armazenada a *string* que contém efetivamente os dados. Se a *string* passada para a função foi um pacote de dados do tipo *broadcast*, é retornado o valor 1, se for do tipo *unicast*, o valor 2. Caso a *string* do argumento não seja definida como um dos dois casos citados, é retornado o código de erro -1.

Para auxiliar na comunicação de dados, o pino de I/O0 do módulo ZigBee é utilizado como indicador. Duas funções foram desenvolvidas para acessar o pino de I/O0, *int zigbee_io_0_local_access (char *status, int x)* para acesso do I/O0 no módulo integrado a placa e, *int zigbee_io_0_remote_access (char *device_id, char *status, int x)* para acesso remoto. A primeira função monta a *string* de controle ATS0F0 = <DATA> e a segunda ATSREMS0F0 = <DATA>, onde DATA é a *string* apontada pelo argumento **status*, que pode ser “0” ou “1”. O argumento *x* de ambas as funções indica quantas tentativas de transmissão serão realizadas, antes de ser retornado o erro de código -1. Para a função que acessa o I/O0 de um módulo remoto, deve ser passado o ponteiro que contém o endereço EUI64 do módulo a ser acessado.

O recebimento do arquivo de configuração é realizado pela função *int receive_config_file ()*. Não é passado nenhum argumento para a função, pois ela acessa as variáveis *char SAMPLE_RATE_CH[8][16]* e *char MEM_BUFFER[16]*, que são declaradas como globais. Ao ser chamada, ela aguarda pelo recebimento do arquivo de configuração, que deve ter a formatação específica citada no item 2.3.2.1. Para cada *string* recebida, a função faz uma verificação nos primeiros caracteres, caso estejam incorretos, a

função retorna o código de erro -1. Se a *string* recebida esta correta, a mesma é armazenada em posição específica da uma matriz de *strings* declarada globalmente. A última *string*, que representa o tamanho do *buffer*, é armazenada na variável *MEM_BUFFER*.

A função *int send_config_file (char *device_id)* realiza o envio do arquivo de configuração, recebido pelo PC, para o módulo com o endereço EUI64, passado como argumento da função. A função acessa as variáveis globais *SAMPLE_RATE_CH[8][16]* e *MEM_BUFFER[16]* e envia-as para o do módulo especificado. Se ocorrer algum erro durante o envio, a função retorna imediatamente com o código de erro -1.

Por fim, a função *void receive_mem_buffer ()*, desenvolvida para receber o *buffer* de memória da placa de transmissão. A placa de transmissão envia os dados dos *buffers* no seguinte formato:

```

UCAST:<EUI64>=CH_0
UCAST:<EUI64>=<PATTERN_1>/<DADO_1>;<DADO_2>;<DADO_3>;...;<DADO_n>;
UCAST:<EUI64>=<PATTERN_2>/<DADO_1>;<DADO_2>;<DADO_3>;...;<DADO_n>;
UCAST:<EUI64>=<PATTERN_1>/<DADO_1>;<DADO_2>;<DADO_3>;...;<DADO_n>;
...
UCAST:<EUI64>=END
UCAST:<EUI64>=CH_1
UCAST:<EUI64>=<PATTERN_1>/<DADO_1>;<DADO_2>;<DADO_3>;...;<DADO_n>;
UCAST:<EUI64>=<PATTERN_2>/<DADO_1>;<DADO_2>;<DADO_3>;...;<DADO_n>;
...
UCAST:<EUI64>=END
...
UCAST:<EUI64>=CH_7
UCAST:<EUI64>=<PATTERN_1>/<DADO_1>;<DADO_2>;<DADO_3>;...;<DADO_n>;
...
UCAST:<EUI64>=END

```

Onde:

PATTERN_1: “0A”;

PATTERN_2: “A0”;

DADO_n: caracteres *ascii* de 0 à 65535

A primeira *string* que a função espera ser recebida é a identificação do canal 0 (CH_0), qualquer outro dado é desprezado. Assim que a função recebe e identifica a *string* “CH_0”, o software receberá os dados do *buffer* de memória, referentes ao canal, até que a *string* “END” seja enviada pela placa de aquisição. O processo se repete até que uma *string*

“END” seja recebida após a leitura dos dados do *buffer* do canal 7.

Apesar de transmissões do tipo *unicast* aguardarem a confirmação do recebimento dos dados, aumentando a confiabilidade do sistema, esta transmissão de confirmação pode falhar e, o pacote pode ser enviado novamente e recebido mais de uma vez. Para evitar este acontecimento, uma *string* conhecida é inserida no início do pacote de dados, denominada *pattern*, na descrição abordada anteriormente. Esta *string* intercala seus valores entre “0A” e “A0” para cada pacote transmitido, portanto, dois pacotes transmitidos em sequência nunca podem apresentar o mesmo *pattern*. Quando dois pacotes, em sequência, são recebidos com o mesmo *pattern*, o último é desprezado, e a rotina permanece esperando um pacote de dados com o *pattern* complementar.

A informação da *string* recebida é separada pela função *str_inf* e, a informação dividida em *sub-strings* pela função de biblioteca *strtok*. São informados à função quais caracteres devem ser substituídos por NULL. Após, são realizadas múltiplas chamadas à função até que todos os ponteiros de todas as *sub-strings* sejam descobertos. Os caracteres escolhidos para realizar a delimitação das *strings* foram ‘/’ e ‘;’.

Neste ponto é verificado o *pattern* da *string* recebida. Se o *pattern* é o esperado, a *string* é enviada pela interface RS-232 ou USB, sendo cada dado, apresentado em uma nova linha.

2.3.2 Software da Placa de Aquisição

O *software* implementado na placa de aquisição não possui interface com o usuário, mas foram incluídos nele, inúmeras rotinas que fazem uso da interface RS-232 e USB para saída de informações de depuração. Estas funções foram propositalmente deixadas para que a execução do *software* possa ser avaliada.

O desempenho da execução do *software* é crítico, para que taxas na ordem de milhares de amostras por segundo sejam alcançadas. Assim, são usadas inúmeras variáveis

globais, acessadas principalmente por funções relacionadas com a aquisição e leitura dos ADCs, desta maneira, economiza-se o tempo para criação das variáveis cada vez que a função é acessada.

2.3.2.1 Execução da Função Principal

A função principal é mostrada em fluxograma nas figuras 15, 16 e 17. Antes da início da execução de instruções de configuração, o *software* aguarda pela estabilização do circuito de PLL do microcontrolador, que demora alguns milissegundos, para que fique sintonizado na frequência de operação.

Configurações costumeiras são as primeiras a serem realizadas: definição das portas de entrada e saída do microcontrolador (*init_trisx()*), valores iniciais para cada porta (*init_ports()*), desabilitação dos ADCs e configurações das duas interfaces UARTs (*baud rate* de 19200, 8 *bits* de comunicação, sem paridade, sem controle de fluxo e 1 *bit* de parada). Em seguida o *software* verifica se existe conexão nas interfaces RS-232 ou USB, mas se não há, seta a interface USB como padrão. Não faz sentido o *software* ficar esperando que uma conexão externa seja realizada, se a placa de aquisição foi concebida para ter mobilidade livre. Uma espera de 5 segundos é inserida neste ponto para que o módulo ZigBee tenha tempo suficiente para realizar sua inicialização.

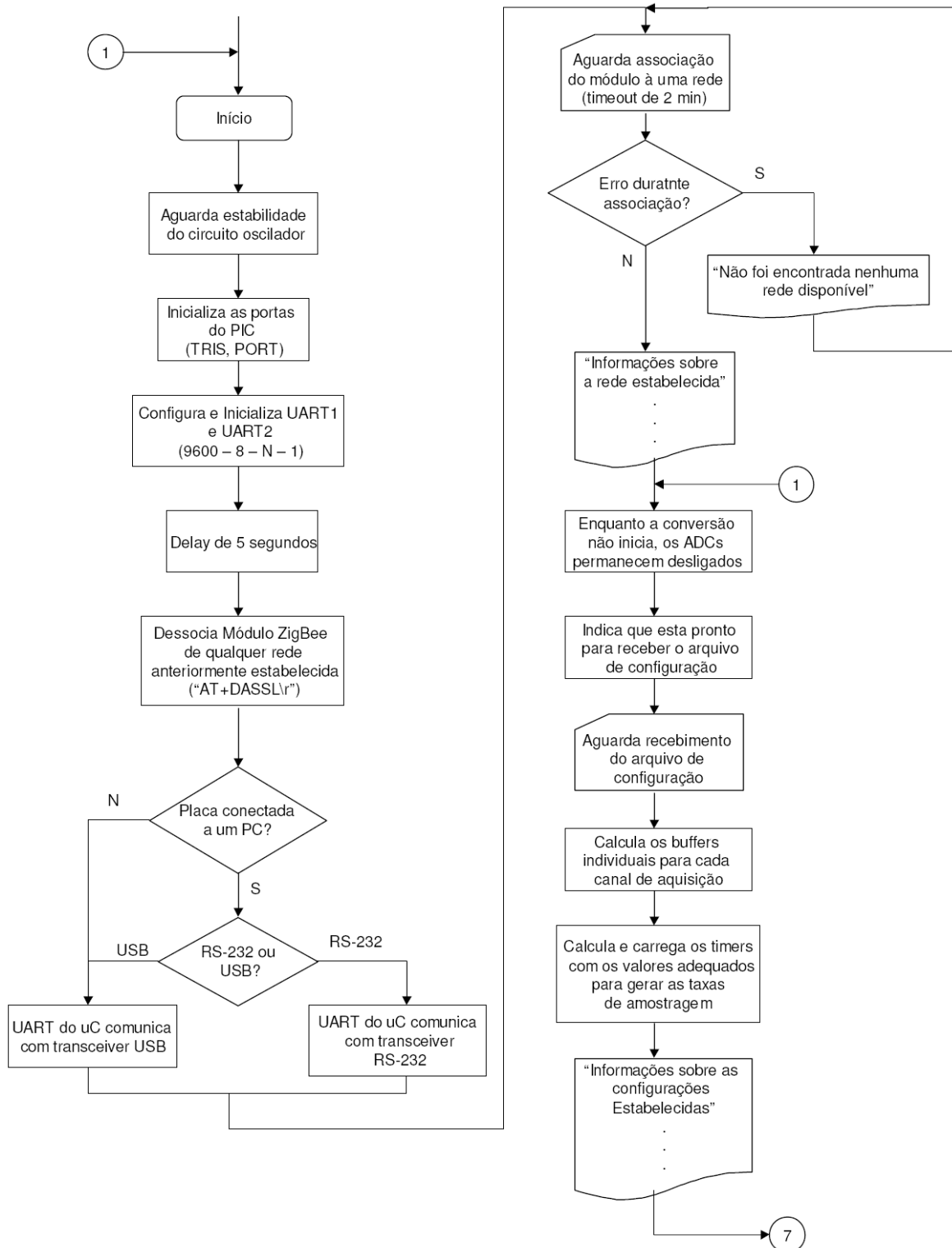


FIGURA 15: FLUXOGRAMA DO SOFTWARE PRINCIPAL DA PLACA DE AQUISIÇÃO

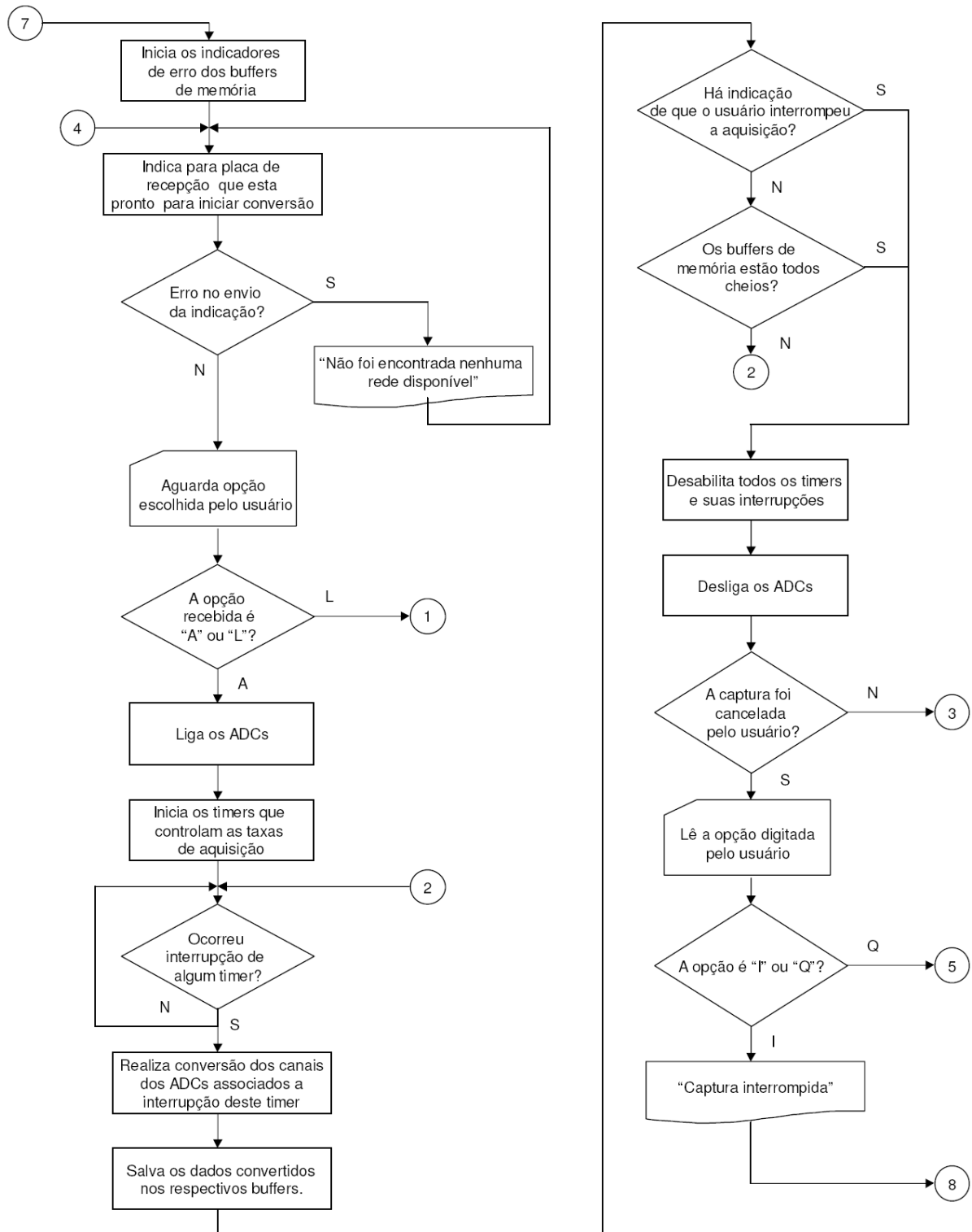


FIGURA 16: FLUXOGRAMA DO SOFTWARE PRINCIPAL DA PLACA DE AQUISIÇÃO

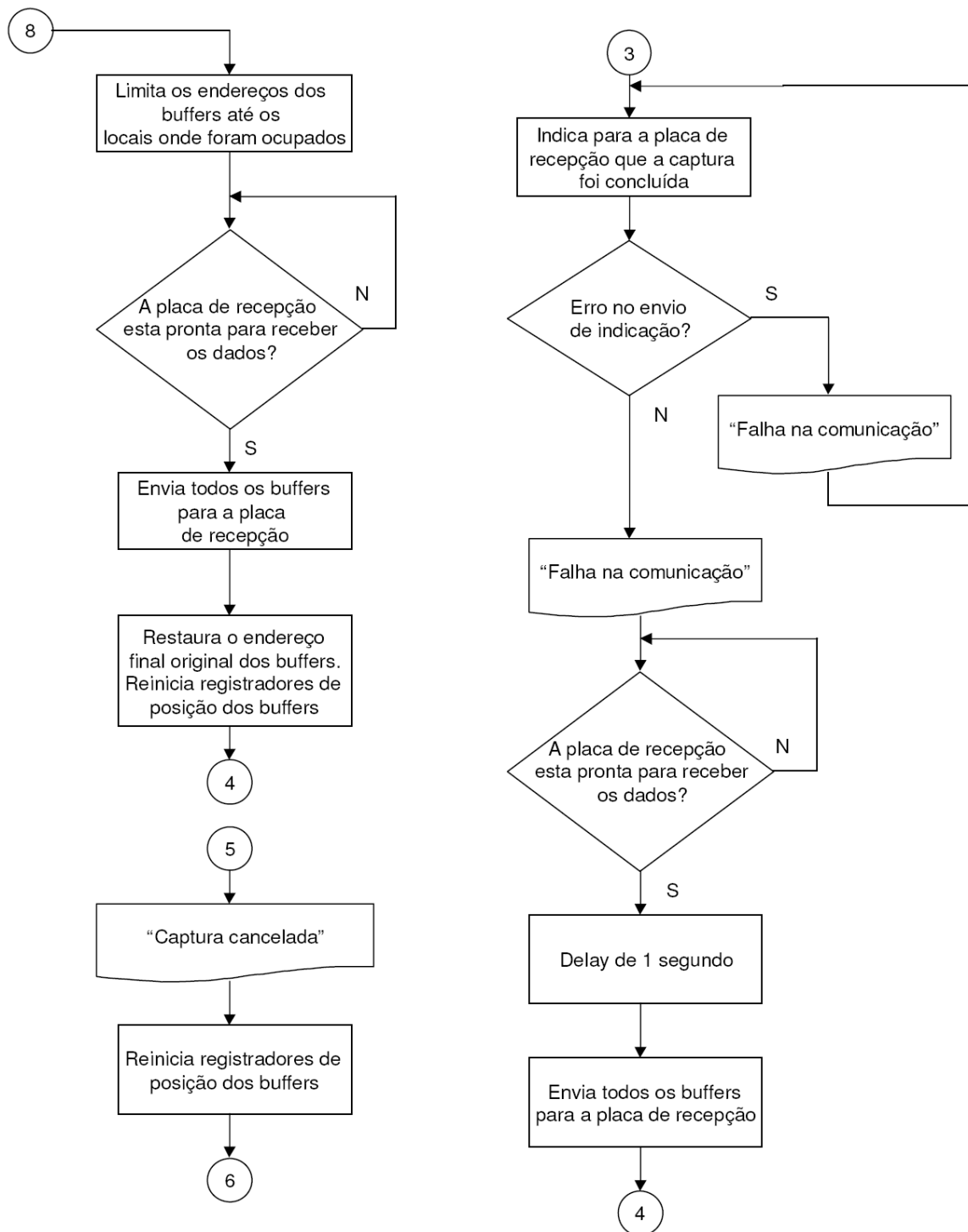


FIGURA 17: FLUXOGRAMA DO SOFTWARE PRINCIPAL DA PLACA DE AQUISIÇÃO

A instrução “AT+DASSL” é passada ao módulo, para que ele desassocie-se de qualquer rede que antes integrasse. Em seguida, o *software* executa a função de associação à rede (*join_WPAN*). Nesta função, as informações de EUI64, nome do nó, número de identificação da rede, canal em que a rede esta estabelecida e EUI64 do *parent* são adquiridas. Ressalta-se que o módulo integrado à placa de aquisição é configurado como RFD, portanto comunica-se apenas com seu *parent*. Finalizando este processo, todas essas informações são enviadas para a interface USB (ou RS-232), apenas para depuração. Durante todo o processo de associação à rede, se ocorrer problemas, informações de erro são enviadas também pela interface UART.

Quando a placa de aquisição associa-se à rede, ela automaticamente envia uma mensagem para a placa de recepção, indicando que esta pronta para receber um arquivo de configuração. Para tornar mais ágil e simples este processo e, não ser necessário a comparação e manipulação constante de *strings*, optou-se por fazer esta indicação apenas escrevendo nível lógico 1 na I/O0 do módulo ZigBee integrado à placa de recepção. Ao receber este sinal, a placa de recepção envia o arquivo para a placa de aquisição. Cada *string* do arquivo de configuração é recebido e conferido pelo *software*. As *strings* são devidamente processadas e transformadas em informação numérica, para realização dos cálculos dos *buffers* e taxas de aquisição, que são realizados pela execução das duas próximas funções. Em seguida são disponibilizados pela interface USB, ou RS-232, as informações do arquivo de configuração, os valores dos *timers* que geram as taxas de aquisição, os preescalares de cada *timer*, endereços de início e fim do *buffer* de memória de cada canal de aquisição.

Realizado o processo de configuração, o *software* prepara a placa para iniciar as aquisições. Indicadores de erro dos *buffers* de memória são iniciados, de forma que os *buffers* de memória dos canais que não são utilizados, são iniciados com indicação de erro, razão que ficará clara mais adiante. Em seguida é enviada indicação à placa de recepção, informando que a placa de aquisição espera por um comando do usuário. São esperadas as opções para início de aquisição “A” ou carga de um novo arquivo de configuração “L”. Se a opção “L” for recebida pelo módulo ZigBee, então todo o processo de espera,

armazenamento, processamento e configuração são executados novamente. Caso a opção “A” seja recebida, os ADCs são ligados (*ADC_mode ('A')*) e os *timers* são iniciados (*start_timers ()*).

As rotinas para realizar os comandos de conversão dos ADCs foram todas escritas dentro de interrupções. Assim, quatro *timers* do microcontrolador controlam as taxas de aquisição dos oito canais disponíveis na placa. A função de inicialização dos *timers* (*start_timers()*), realiza a habilitação das interrupções dos *timers* que controlam as taxas de aquisição dos canais que estão ativos. Os canais de aquisição CH_0 e CH_4, CH_1 e CH_5, CH_2 e CH_6, CH_3 e CH_7 são respectivamente controlados pelos *timers* 2,3,4 e 5 do microcontrolador. Para que quatro canais possam ser adquiridos simultaneamente, dois *timers* são disparados por vez. Os *timers* 2 e 3 são disparados juntos e, cerca de 1000 ciclos de máquina depois, são disparados os *timers* 4 e 5. Uma ressalva deve ser feita, nos *timers* que são disparados em conjunto, há diferença de 1 ciclo de máquina do microcontrolador, ou seja, cerca de 62,5ns.

Para tornar o sistema mais flexível, canais que estão associados a um *timer* podem ter taxa de aquisição diferente, desde que, as taxas de aquisição entre canais sejam múltiplas da base de tempo do *timer*. Por exemplo, se a taxa de aquisição do CH_0 for de 1000sps (1/1ms), as taxas do CH_4 podem ser 1000sps (1/1ms), 500sps (1/2ms), 333,33sps (1/3ms), 250sps (1/4ms), 200sps (1/5ms) e assim por diante.

Após a inicialização dos *timers*, o *software* fica aguardando o acontecimento de uma interrupção, em uma linha com a instrução “*while (!interrupt);*”. No atendimento de uma interrupção, são indicados quais os canais foram convertidos e, a variável *interrupt* é carregada com o valor da interrupção que aconteceu (de 1 à 5), desta forma o *software* principal sai do laço *while* e continua sua execução. A próxima função executada (*save_adc_data_to_buffers*) realiza a leitura e armazenamento dos canais dos ADCs convertidos pela interrupção. Finalizado este processo, é verificado, lendo-se o pino de I/O0 do módulo ZigBee, se não houve requisição do usuário para interrupção ou cancelamento do ensaio. A condição para término das aquisições depende do estado dos *buffers* de memória e do indicador de intervenção do usuário. Se todos os *buffers* de memória

estiverem cheios ou, ocorreu uma intervenção do usuário requisitando a interrupção ou cancelamento das aquisições, o processo é finalizado e o *software* passa para a próxima etapa.

Terminado o processo de aquisição, todos os *timers* são desligados e suas interrupções desabilitadas. Se o processo de aquisição foi terminado pela intervenção do usuário, a módulo ZigBee espera a transmissão da opção que foi requisitada pelo mesmo. Duas opções são esperadas: “I”, indicando que o processo foi interrompido e “Q”, indicando que o processo foi cancelado. No caso do cancelamento, nenhum dado de aquisição é transmitido para a placa de recepção e, a placa de aquisição aguarda por um novo comando de “A” aquisição ou “L” para carregar novas instruções de configuração. Caso a opção informada foi para interromper a aquisição, os dados capturados até o momento da intervenção são enviados para a placa de recepção. No último caso, onde não há mais espaço suficiente nos *buffers*, todo o espaço de memória é transmitido.

2.3.2.2 Descrição das Funções Implementadas

As funções *unicast* (), *broadcast* (), *read_string_uart_pc* (), *zigbee_io_0_local_access* (), *zigbee_io_0_remote_access* () e *str_inf* () são comuns a placa de recepção, portanto suas descrições não serão abordadas aqui.

A função *int join_WPAN* (*char *device_id*, *char *wpan_ch*, *char *wpan_id*, *char *parent_id*) é responsável por obter informações sobre a rede em que o módulo ZigBee irá associar-se. O processo de associação, que ocorre a cada um minuto, é realizado automaticamente pelo módulo quando ele não faz parte de nenhuma rede. Os argumentos passados para a função são ponteiros para *strings* que irão armazenar, respectivamente, o EUI64 do módulo, o número de identificação da rede, o canal onde ela será estabelecida e o EUI64 do *parent* com o qual o módulo irá comunicar-se. O EUI64 do módulo é obtido pelo envio do comando “ATI”, que irá retornar também outras informações que não são usadas pelo *software* de aplicação. As informações sobre a rede são automaticamente passadas do

módulo para o microcontrolador, assim que a rede for estabelecida. A identificação do *parent* é obtida pelo envio do comando “AT+PARENT?”, que retorna então o EUI64 do *parent* conhecido pelo módulo integrado à placa de aquisição.

Para recepção das informações de configuração, foi desenvolvida a função *int receive_config_file (float sample_rate_ch[8], long *sram_size)*. Esta função recebe cada *string* do arquivo de configuração (uma linha do arquivo) e, separa os campos de identificação (CH_x ou BUFFER) dos campos que representam os valores numéricos. Utiliza para tal, a função de biblioteca *strtok*, com o caractere de tabulação (“\t”) como argumento, para realizar a separação das *strings*. Lembra-se que arquivo de configuração foi descrito no item 2.3.2.1. Após a separação dos campos da *string*, a função confere o campo de identificação. Se a *string* de identificação for diferente da esperada, a função descarta as informações recebidas e espera pela recepção de uma nova *string*. Se a *string* recebida é a correta, o campo da *string* que representa o valor numérico é armazenado em uma posição da matriz, declarada no início da função. Este processo de receber, comparar e armazenar, se repete até que a última linha do arquivo de configuração, que contém o campo de identificação “BUFFER”, seja recebido. Após todos os dados recebidos, as *strings* são convertidas para valores numéricos, para que possam ser manipuladas matematicamente por outras funções do *software*, e armazenadas nas variáveis passadas com argumento da função.

Para da determinação do tamanho do *buffer* de memória de cada canal de aquisição, foi implementada a função *void init_mem_buffer_regs (void)*. Esta função acessa inúmeras variáveis globais para realizar os cálculos e armazenar os resultados. As variáveis de entrada são a matriz *SAMPLE_RATE_CH[8]*, que contém a taxa de amostragem de cada canal de aquisição em amostras por segundo, e *MEM_BUFFER*, que representa o quanto de memória, em *words*, foi disponibilizado para armazenamento. Para o cálculo do tamanho de cada *buffer*, é aplicada a equação:

$$BUFFER_CH[x] = \frac{MEM_BUFFER \cdot SAMPLE_RATE_CH[x]}{\sum_{i=0}^7 SAMPLE_RATE_CH[i]}$$

Como o resultado do cálculo é em ponto flutuante, somente o valor inteiro é aproveitado. Em seguida são calculados os endereços iniciais e finais de cada *buffer*. Os resultados são armazenados nas matrizes globais MEM_ADDR_BOT_CH[8], para os endereços iniciais, e MEM_ADDR_TOP_CH[8], para os endereços finais. Os *buffers* são dispostos em sequência, sendo que, onde termina o endereço de um (TOP_CH[x]), começa o endereço do outro (BOT_CH[x+1]+1). Uma matriz chamada MEM_ADDR_ACT_CH[8], que representa as posições de endereço atual dos *buffers*, tem seus valores iniciados com o endereço base de seus respectivos *buffers*.

As taxas de amostragem, que são fornecidas pelo usuário em amostras por segundo, são convertidas em valores específicos para carga nos *timers* pela função *void define_sample_rates (void)*. A primeira operação que a função realiza é transformar os valores das variáveis globais SAMPLE_RATE_CH[8] em segundos. Para isto a simples operação de inversão é realizada. Os valores, agora em segundos, são armazenados nas próprias variáveis SAMPLE_RATE_CH[8]. Devido a cada *timer* do microcontrolador controlar a taxa de aquisição de dois canais e, o sistema projetado para permitir que ambos os canais tenham taxas de amostragem diferentes, desde que múltiplas da base de tempo do *timer*, a rotina realiza uma comparação para determinar qual dos canais controlados pelo *timer* possui a maior e a menor taxa de amostragem. O canal que possuir a maior taxa de amostragem determinará qual a base de tempo do *timer*. Na realidade, ambos os canais são convertidos quando há o atendimento à rotina, pois o ADC amostra dois canais por vez. Mas pode-se ler o primeiro canal, sempre que a interrupção acontece e, ler o segundo em uma taxa menor. Por exemplo, ler o segundo canal na terceira leitura do primeiro canal. Desta forma caracteriza-se uma segunda taxa de amostragem, que é relação múltipla da base de tempo do *timer*. Para realizar este controle, utiliza-se a variável SR_REL[8], que contém as relações entre as bases de tempo dos *timers* e as taxas de amostragem de cada canal. Por exemplo, CH_0 com taxa de amostragem de 1000sps e CH_4 com taxa de amostragem de 500sps. Estes dois canais são controlados pelo *timer* 2 do microcontrolador. O valor de SR_REL[0], seria neste caso 1, pois o CH_0 tem a maior taxa de amostragem, portanto o *timer* 2 gerará uma interrupção a cada 1ms. SR_CH[4] teria o valor 2, pois a taxa de amostragem é duas vezes menor que a base de tempo do *timer*.

Para gerar taxas de amostragem mais lentas, configuram-se os registradores preescalares dos *timers*. O microcontrolador possui a opção de dividir o tempo de incremento dos *timers* em 8, 64 e 256. A função identifica qual o preescalar que deve ser configurado nos registradores do microcontrolador, para cada valor de taxa de amostragem informado.

Para que a placa de aquisição possa iniciar a captura de dados, foi escrita a função *void start_timers (void)*. Esta função tem funcionamento bastante simples. A função verifica a variável *SAMPLE_RATE_CH[x]* e habilita a interrupção do *timer* correspondente se seu valor for diferente de zero. Após habilitar as interrupções necessárias, os *timers* 2 e 3 são disparados e cerca de 1000 ciclos de máquina depois, os *timers* 4 e 5. Este tempo de disparo entre os pares de *timers*, é necessário para que o sistema possa atender a interrupção gerada por um par de *timers*, e executar todas as instruções subseqüentes, sem que ocorra a interrupção do outro par de *timers*, o que poderia comprometer as taxas de amostragem do sistema.

Para leitura de dois canais do ADC amostrados simultaneamente, foram desenvolvidas duas funções: *void read_2_channels_ADC_0 (unsigned char x, unsigned char y)* e *void read_2_channels_ADC_1 (unsigned char x, unsigned char y)*. Cada uma destas funções controla a leitura de um circuito integrado ADC da placa de aquisição. São passadas como argumento da função os números dos canais que se deseja ler. A função faz uso de duas variáveis globais: *ADC_VALUE_CH[8]* e *CH[8]*. A primeira variável é destinada a armazenar, em local específico da matriz, o valor adquirido por cada canal. *ADC_VALUE_CH[0]*, armazena as conversões do *CH_0*, *ADC_VALUE_CH[1]* do *CH_1* e assim por diante. A variável *CH[8]* serve como um *flag* para indicar se o canal, que pode ser identificado pelo índice da matriz, deve ser lido e armazenado nos *buffers* de memória. O valor 1 indica que a operação deve ser realizada.

Para que o tempo de execução da função fosse bastante reduzido, algumas limitações foram impostas. Os argumentos podem ser 0,4 ou 1,5, nesta ordem específica, para a função que controla o *ADC_0*. Para a função que controla o *ADC_1*, os argumentos podem ser 2,6 ou 3,7. Isto se faz necessário, devido a ordem como os canais são avaliados

dentro da função e, as posições de memória onde os dados adquiridos são salvos.

As primeiras instruções da função habilitam o barramento do ADC para comunicar-se com o microcontrolador. Em seguida, as variáveis CH[x] e CH[y], são avaliadas, para verificar se o canal será armazenado temporariamente nas variáveis globais ADC_VALUE_CH[x] e ADC_VALUE_CH[y]. Se verdadeiro, cada condicional *if* seleciona um dos canais para ler e armazenar. Por fim o barramento do ADC é desabilitado.

A execução da função *void read_4_channels_ADCs (unsigned char x, unsigned char y, unsigned char z, unsigned char w)* é bastante similar, com a diferença de que ela realiza a leitura de até quatro canais dos ADCs (dois canais do ADC_0 e dois canais do ADC_1). As restrições e observações sobre a ordem e tipos de argumentos são similares também. Só podem ser passados os argumentos 0,4,2,6 ou 1,5,3,7 nesta ordem específica.

Para salvar os valores das conversões AD, foi desenvolvida a função *int write_SRAM (unsigned char x)*. Esta função faz acesso as variáveis globais MEM_ADDR_BOT_CH[8], MEM_ADDR_ACT_CH[8] e MEM_ADDR_TOP_CH[8]. Respectivamente, essas variáveis informam, o endereço inicial do *buffer* de memória do canal representado pelo índice da variável, o endereço atual e o endereço final do *buffer*. Também é acessada a variável ADC_VALUE_CH[8], que representa a matriz, localizada na memória SRAM do microcontrolador, que contém os valores de conversão AD de cada canal da placa de aquisição.

O argumento passado na chamada da função indica qual canal de aquisição deve ser lido. Este argumento é passado diretamente para os índices de todas as variáveis citadas. Ao iniciar a execução da função, realiza-se uma verificação de limites nos endereços dos *buffers* de memória. Se o endereço atual for maior que o endereço final, a função retorna imediatamente com o código de erro -1. Caso o endereço atual e o final sejam iguais, a função é executada normalmente, e ao final é retornado pela função o valor 0, indicando que a última posição do *buffer* foi ocupada. Se ainda houver espaço no *buffer*, a função retorna com o valor 1. A execução da função é simples. Primeiramente, disponibiliza-se o endereço atual do *buffer* no barramento de endereços da placa. A memória SRAM é então habilitada por seu *chip select*. Em seguida o valor da variável ADC_VALUE_CH[x] é

escrito no barramento de dados e, depois enviado um pulso de escrita ao pino da memória SRAM. Ao final, a memória SRAM é desabilitada e, a variável `MEM_ADDR_ACT_CH[x]` incrementada, apontando o *buffer* para o próximo endereço disponível. Uma função similar, *int read_SRAM (unsigned char x)*, faz a leitura da memória SRAM e armazena os dados lidos na variável `ADC_VALUE_CH[x]`.

Após uma interrupção ser atendida, a função *void save_adc_data_to_buffers (unsigned char CHANNEL)* é executada na sequência. Ela é responsável pela leitura e armazenamento dos dados obtidos das conversões AD. Para tanto, *read_2_channels_ADC_0 ()*, *read_2_channels_ADC_1 ()* e *write_SRAM ()* fazem parte integrante do código da função. O argumento passado na chamada da função é um valor, carregado durante o atendimento da interrupção, que indicam quais os canais devem ser avaliados para leitura e armazenamento. A função faz acesso direto às variáveis globais `SR_REL[8]`, `SR_REL_I[8]`, `CH[8]` e `BUFFER_ERROR[8]`. Os índices das matrizes, de 0 à 7, informam sobre qual canal a posição da matriz faz referência. Índice 0 corresponde `CH_0`, índice 1 corresponde `CH_1`, e assim por diante. Respectivamente, estas variáveis informam: a relação entre a base de tempo da taxa de amostragem e a base de tempo do *timer*; quantas vezes o *timer* que controla a taxa de amostragem estourou; indica, pela escrita do valor 1, que o canal de ser lido na função *read_SRAM ()* e salvo nos *buffers* de memória pela função *write_SRAM ()*; indica se o *buffer* de memória ainda possui espaço para armazenamento.

A função é implementada com um *switch case* que avalia seis casos: leitura e armazenamento dos canais 0 e 4; 1 e 5; 2 e 6; 3 e 7; 0, 2, 4, 6 ou 1, 3, 5, 7. A execução da função é bastante similar nos seis casos, ela apenas faz referências a canais diferentes em cada caso.

O primeiro passo da função é descobrir qual dos canais do caso, possui a maior taxa de amostragem. Para isto é implementado um teste *if* na variável `SR_REL[x]`. Se `SR_REL[x]` for valor 1, este é o canal que possui a maior taxa de amostragem. Em seguida, é acionado o registrador que serve de *flag*, `CH[x]`, para indicar que o canal deve ser armazenado. O segundo canal, `SR_REL[x + 4]` é testado para verificar se esta sendo

utilizado pelo usuário. Qualquer valor inteiro diferente de 0, indica que o canal está ativo. Se o canal está ativo, a variável `SR_REL_I[x + 4]` é incrementada e em seguida testada. Quando o valor de `SR_REL_I[x + 4]` for igual ao de `SR+REL[x + 4]`, o *flag* `CH[x + 4]` será acionado, indicando que o canal deve ser armazenado. Com esta estrutura de *software* realizamos as taxas de amostragem diferentes para dois canais controlados por um mesmo *timer*. Depois destes testes condicionais, é chamada a função `read_2_channels_ADC_0 (x, x+4)`. Os dois canais são sempre lidos, mas são armazenados nos *buffers* de memória apenas se o *flag* `CH[x]` do canal estiver acionado, caso contrário o valor lido é descartado. O armazenamento nos *buffers* é realizado pela chamada da função `write_SRAM(x)`. O valor de retorno desta função é sempre avaliado, sendo que, quando retornado um erro, -1, este erro também será retornado por `save_adc_data_to_buffers`. Toda esta estrutura discutida é similarmente implementada nos outros 5 casos avaliados dentro da função.

Para enviar dos *buffers* de memória, para o terminal remoto, em um formato específico, foi desenvolvida a função `void send_mem_buffer ()`. A função faz uso das variáveis globais `parent_id`, `MEM_ADDR_ACT_CH[8]`, `MEM_ADDR_BOT_CH[8]`, `ADC_VALUE_CH[8]`, e também faz chamada às funções `unicast` e `int_to_string`.

A função é implementada em um laço *for*, que se repete oito vezes, para enviar o *buffer* dos oito canais da placa de aquisição. O primeiro passo da função é enviar à placa de recepção, a identificação do canal do qual o *buffer* que será transmitido faz parte. A identificação do canal é uma simples *string* no formato “CH_x”, onde *x* representa o número do canal. É realizado um teste condicional, através da variável `SR_REL[x]`, para verificar se o canal foi utilizado nos ensaios. Se o canal foi utilizado, `SR_REL[x]` diferente de zero, então a transmissão dos dados de captura é iniciada, caso contrário, a *string* “END” é enviada à placa de recepção, indicando o término da transmissão dos dados pertencentes ao canal.

Cada pacote de dados do módulo ZigBee pode conter até 65 *bytes*. A função é implementada de forma a utilizar o máximo de ocupação do pacote, para aumentar o desempenho da taxa de transmissão do sistema. O primeiro dado do pacote é um *pattern*, que alterna entre “0A” e “A0” para cada transmissão. Dois pacotes transmitidos em

seqüência nunca possuem o mesmo *pattern*, isto é realizado por motivos já discutidos no item 2.3.2.2. Em seguida um caractere de separação “/” é inserido. Após, os dados são lidos da SRAM, pela função *read_SRAM*, convertido em *string* pela função *int_to_string* e concatenados ao pacote pela função de biblioteca *strcat*. Um caractere de “;” separa cada dado inserido ao pacote. Quando o pacote termina de ser montado, ele é imediatamente enviado à placa de recepção. Este laço de *software* se repete até que todo o *buffer* de memória do canal seja transmitido e, ao final, a *string* “END” é transmitida, indicando o último dado.

Quatro interrupções de *timers* realizam o controle das conversões AD. Elas são declaradas no código, com a seguinte sintaxe:

```
void __attribute__((__interrupt__(__preprologue__("DISI #900")))) _T2Interrupt (void);
void __attribute__((__interrupt__(__preprologue__("DISI #900")))) _T3Interrupt (void);
void __attribute__((__interrupt__(__preprologue__("DISI #900")))) _T4Interrupt (void);
void __attribute__((__interrupt__(__preprologue__("DISI #900")))) _T5Interrupt (void);
```

Respectivamente, elas representam as interrupções dos *timers* 2, 3, 4 e 5. Esta é a sintaxe utilizada pelo compilador C30, para declarar o código que será executado pela interrupção. É utilizado na declaração, um parâmetro opcional chamado *preprologue*. Este parâmetro informa à interrupção, alguns comandos em *assembly* que devem ser executados assim que ocorre a indicação da interrupção no *hardware* do microcontrolador, ou seja, estas instruções são executadas antes do próprio prólogo, que antecede a execução do código a ser executado pela interrupção. Entende-se como prólogo, as instruções que são executadas para a preparação do atendimento à interrupção. O comando “DISI #900”, desabilita as interrupções do microcontrolador durante 900 ciclos de máquina, tempo suficiente para que todos os processos posteriores ao atendimento à interrupção (ler ADCs, salvar dados nos *buffers* de memória, etc.) possam ser executados sem que outra interrupção ocorra.

O código que é executado pela interrupção, apenas faz a seleção do canal do ADC a ser convertido e realiza o pulso de conversão. Após, aguarda até que o processo de conversão termine. A variável CHANNEL é carregada com o código dos canais que foram convertidos e que serão avaliados na função *save_adc_data_to_buffers*. Cada interrupção,

também faz a verificação do *flag* de interrupção do outro *timer* que foi disparado junto com o seu (ver função *start_timers*). A interrupção do *timer 2* verifica o *flag* do *timer 4* e vice-versa. Assim como a interrupção do *timer 3* faz a verificação do *flag* do *timer 5* e vice-versa. Se no atendimento da interrupção de um *timer*, o *flag* do outro *timer* esta acionado, quatro canais são convertidos simultaneamente.

3 DISCUSSÃO DOS RESULTADOS

Projetos que envolvem *hardware* e *software* integrados costumam demandar um tempo de depuração razoável. Nem sempre é fácil identificar se determinado comportamento discrepante é causado por um ou outro e, um processo adequado de testes e validação deve ser implementado. Nos próximos itens serão apresentados alguns testes aos quais foram submetidos o *hardware* e *software* implementados no projeto e, algumas das suas características e limitações.

3.1 Projeto de Hardware

O processo de testes de ambas as placas foi realizado em pequenos blocos. Durante a montagem e integração dos componentes às placas de circuito impresso, cada bloco, representado no Anexo 1, foi montado e em seguida testado. Os primeiros blocos integrados e testados foram as fontes de tensão. As saídas de cada fonte foram devidamente medidas com um multímetro, para verificar se o nível de tensão adequado estava sendo regulado pelos circuitos. Seguindo o processo, foram integrados à placa o bloco da CPU e os conectores CN2 e CN4, que dão acesso direto a alguns pinos do microcontrolador. Escreveu-se então o primeiro *software* de teste, que alternava o nível lógico 1 e 0 em um pino do conector CN2. Este foi o processo de *start-up* de ambas as placas e, a partir daí,

para cada novo bloco integrado à qualquer uma das placas, um pequeno *software* de teste era escrito para a validação do funcionamento do *hardware*. Os três principais blocos de *hardware* que passaram ainda por testes e medições foram as fontes, os conversores ADCs e o módulo ZigBee.

3.1.1 Fontes

O circuito das fontes foi projetado para trabalhar numa ampla faixa de tensão e suportar inversões de polaridades. Seguem nos itens seguintes alguns testes que foram aplicadas a ambas as placas.

Faixa de operação de tensão: os reguladores escolhidos para o projeto das fontes, têm uma característica comercial denominada “*low dropout*”. Este termo é aplicado a reguladores que tem baixa queda de tensão entre seus terminais de entrada e saída, portanto, podem operar com tensões na entrada bastante próximas da tensão nominal de saída. O pré-regulador das placas, LT1085CM (U12), é um regulador que tem queda de tensão inferior a 1V, podendo chegar a 1,5V quando em carga plena de 3A. A tensão de entrada, em relação a tensão de saída, pode chegar até 30V. Dada essas características e, lembrando que a tensão de saída foi projetada para um valor próximo a 6V, discutida no item 2.2.1, foi realizado um teste para determinar a faixa de operação do projeto. O circuito foi testado na faixa de tensão de 7,5V à 28V, não havendo restrição alguma. Para tensões inferiores a 7,5V, o circuito ainda opera, mas começa a haver o descaimento da tensão de saída do regulador. Como foram montadas somente duas placas, tensões acima de 28V não foram testadas para manter a integridade do circuito.

Consumo de corrente: como os reguladores de tensão das fontes são lineares, o consumo de corrente não sofre grandes desvios dentro de toda a faixa de tensão testada. Em ambas as placas, o componente que mais consome corrente é o módulo ZigBee e, este consumo varia bastante quando o módulo esta realizando transmissões de dados. Foram realizadas medidas de corrente através de um resistor *shunt*, conectado na entrada da placa,

e visualizada a forma de onda da corrente em um osciloscópio. Na inicialização, a corrente de ambas as placas varia muito, oscilando entre valores de 20mA à 100mA, mas depois se estabiliza. Para a placa de recepção, a corrente permanece num valor de aproximadamente 70mA, tanto nas transmissões quanto nas recepções. Lembramos que o módulo ZigBee integrado a esta placa é configurado como um coordenador, portanto seu circuito de RF fica permanentemente ligado. Na placa de aquisição, o módulo ZigBee é configurado como um RFD, seu consumo é portanto reduzido. O consumo de corrente da placa, enquanto o módulo esta inoperante, é de aproximadamente 70mA também. Quando são iniciadas transmissões, o consumo da placa sofre picos de corrente de 100mA. Estes picos são causados pelo módulo, que liga seu transceiver de RF somente durante as transmissões. Quando o módulo esta inoperante, estes picos ocorrem a cada segundo, pois ele fica constantemente verificando, junto ao *parent*, se não existem dados que são de seu interesse.

Inversão de polaridade: o circuito foi alimentado com polaridade inversa, em toda a faixa de tensão de operação, de 7,5V a 28V. Na tensão de -28V permaneceu durante 30 minutos e, a corrente medida na entrada foi inferior a 100uA. Ao final do teste, o circuito foi alimentado com a polaridade correta e sua operação ocorreu sem problemas.

Dissipação de potência: o componente que tem a maior dissipação de potência, devido a variação de tensão na sua entrada, é o pré-regulador. Quando as fontes foram projetadas, estimou-se que uma corrente inferior a 250mA seria consumida. Mas mesmo assim, foi selecionado um regulador que suporta corrente de 3A, indiscutivelmente muito maior que o necessário. Essa escolha é justificada pelo fato de que reguladores com correntes de saída pequena têm, em geral, baixa capacidade para dissipação de potência e estreita faixa de operação para tensão de entrada. Na aplicação desenvolvida, a potência dissipada no componente para uma tensão na entrada de 28V e consumo de corrente de 100mA é de aproximadamente 2,1W. Este é um valor moderado para um componente de tamanho reduzido. Seguindo algumas recomendações do fabricante sobre a montagem do regulador sobre uma área de cobre com área de 323mm², uma resistência térmica de aproximadamente 30°C/W (junção-ambiente) é conseguida. Assim, a temperatura na junção, para este caso, teria um acréscimo de aproximadamente 65°C em relação a

temperatura ambiente. Testes em bancada resultaram, para dissipação de 2,1W, um acréscimo de 58°C na junção em relação a temperatura ambiente.

3.1.2 Conversores ADCs

Sistemas de conversão ideais não possuem ruídos, o que na prática não ocorre. Conversores com resolução de 16 *bits* possuem uma discretização de tensão bastante pequena. Para uma faixa de tensão de entrada entre 0 à 5V, um LSB do ADC equivale a 76,294μV. Esta tensão é muito próxima da amplitude do ruído típico garantido pelos reguladores de tensão que alimentam o ADC, que é de 20μV. O consumo de corrente de outros componentes e, do próprio ADC, acabam por agregar ruídos a placa, principalmente se a corrente é impulsiva, pois mudanças de cargas impulsivas nas saídas dos reguladores costumam produzir um pequeno transiente de tensão, que pode ser caracterizado como ruído. Reguladores de tensão também são sensíveis a ruídos na sua entrada. Os reguladores que alimentam os ADCs possuem rejeição de *ripple* típico de 65dB.

Dois testes foram realizados, para verificação dos ADCs. O primeiro teste foi a simples verificação da faixa de tensão de operação na entrada de cada canal. Os resultados dos ensaios comprovaram o funcionamento correto dos canais CH_4, CH_5, CH_6 e CH_7, para toda a faixa de tensão (0V à 5V), sendo linear a relação entre a tensão nas entradas analógicas e o código gerado no barramento de dados do ADC. Os canais CH_0, CH_1, CH_2 e CH_3, apesar de funcionarem bem até o fundo de escala positivo (+5V), não apresentaram bom desempenho quando a tensão era próxima de 0V. Ao colocar a tensão 0V nas entradas destes canais, era convertido pelo ADC um valor, em código decimal, entre 80 e 130. Isto equivale a uma tensão de quase 10mV. Resalta-se que o AD7654 possui quatro canais, sendo dois convertidos simultaneamente por vez. O ADC, de fato, quando é instruído a fazer uma conversão, amostra a tensão dos dois canais e armazena-as em um circuito *sample and hold*, para posterior conversão. No sistema implementado, os canais de CH_0 à CH_3 são os primeiros a serem convertidos. Casualmente são os canais que

apresentaram a não linearidade nas tensões próximas de 0V. Não se tem certeza do porque este problema ocorre, ele ainda esta sendo investigado. Analisando cautelosamente o projeto, levantou-se a hipótese de que o problema está relacionado com o pino que dispara o processo de conversão do ADC. Este é o único pino, relacionado com os circuitos analógicos internos do ADC, que é diretamente conectado ao microcontrolador sem nenhum circuito de interface. A falta de um circuito de interface, isolando este pino dos circuitos digitais da placa, pode estar inserindo ruídos durante o processo de conversão dos canais CH0 à CH3.

Para o segundo teste, utilizou-se uma bateria comercial, formato AAA, com tensão nominal de 1,2V. Foram realizadas 10000 medidas em cada canal, a uma taxa de 1000 amostras por segundo, e realizado um histograma do ensaio. A partir destes dados, levantou-se a média e desvio padrão das medidas. São apresentados nas figuras de 18 a 25 os histogramas do ensaio de cada canal. Devido aos problemas já citados nos canais CH_0 à CH_3, em uma tentativa de depurá-lo, um dos ADCs modelo AD7654 foi trocado por um AD7655. Estes ADCs são pino a pino compatíveis, com a diferença básica de que o AD7655 possui a capacidade de realizar amostragens em até 1Msps. Os canais que são ligados ao AD7655 são CH_0, CH_1, CH_4 e CH_5.

Os histogramas foram limitados para apresentar 50 códigos decimais. Nas colunas de cada extremidade dos gráficos, são acumulados os resultados que estão fora da escala. A tensão da bateria, medida com um multímetro, tem valor de 1,453V.

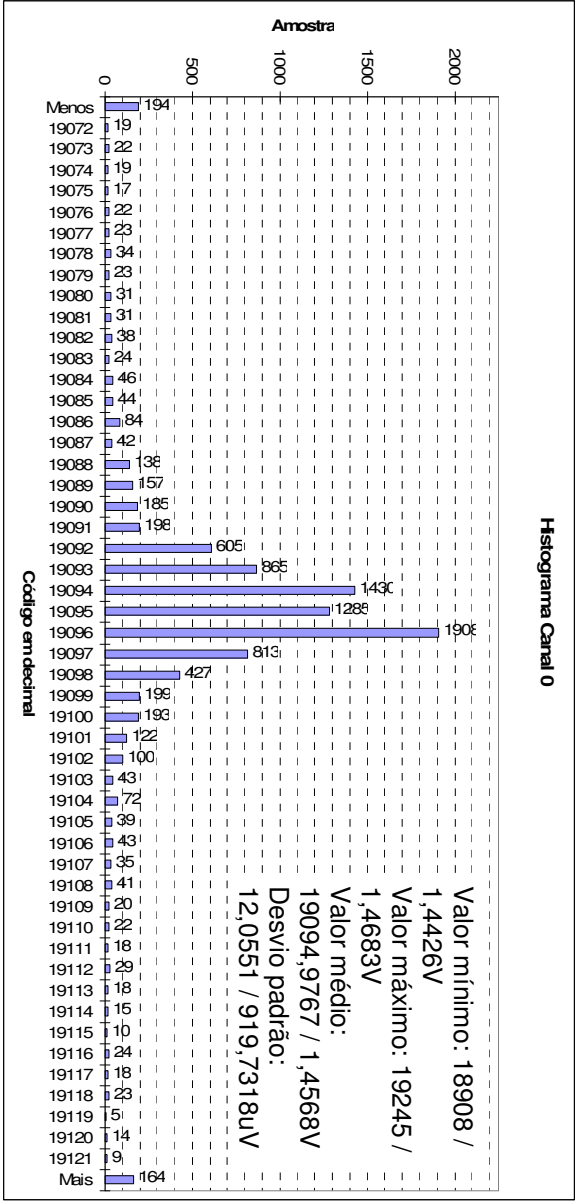


FIGURA 18 - HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 0

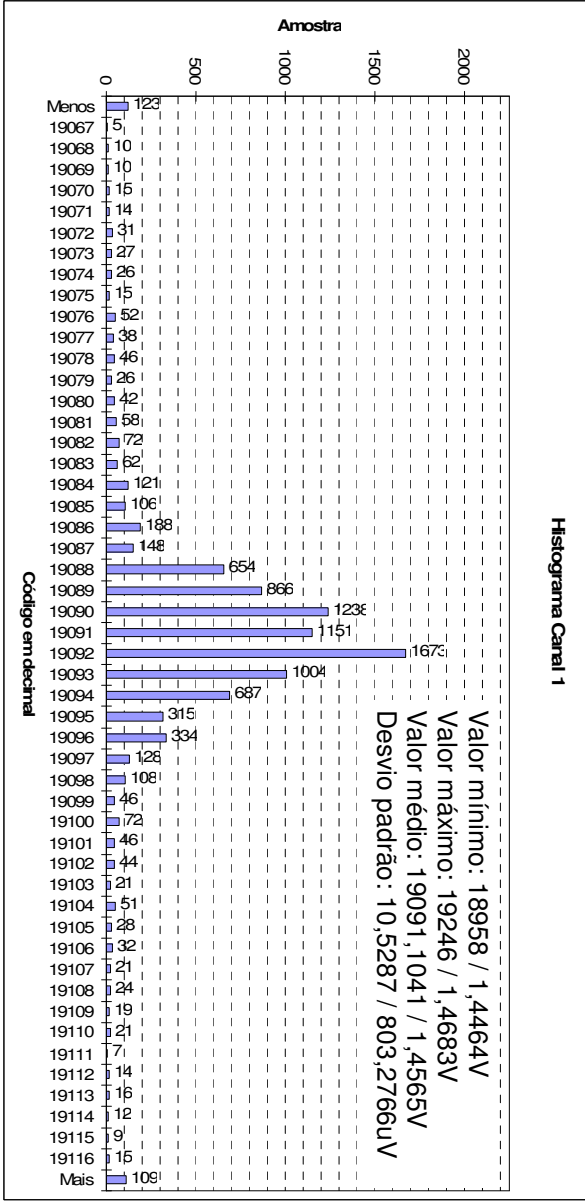


FIGURA 19 - HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 1

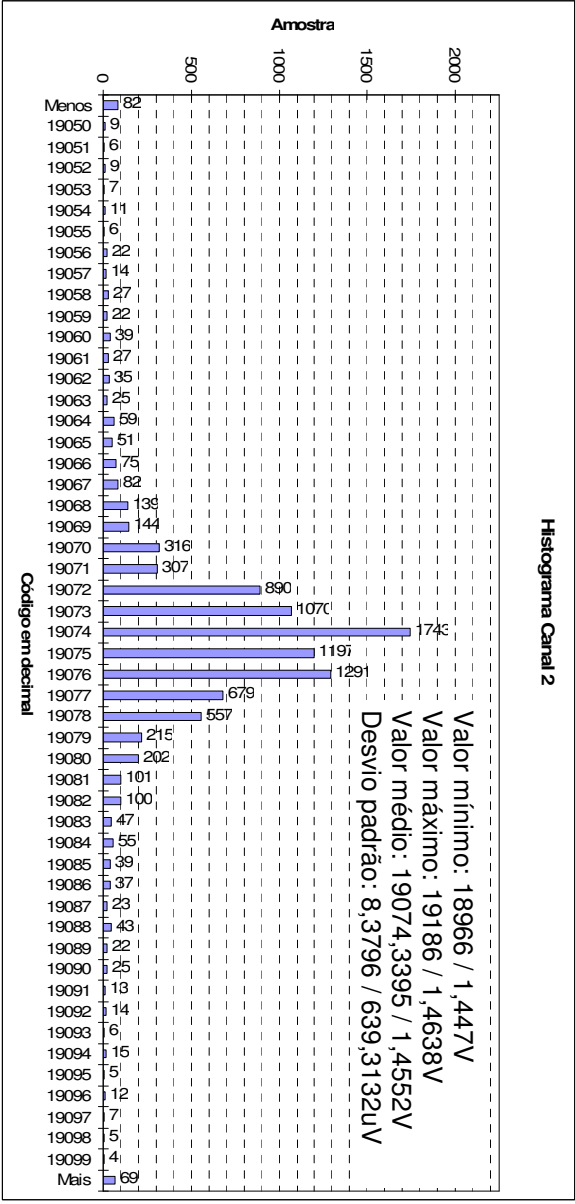


FIGURA 20 - HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 2

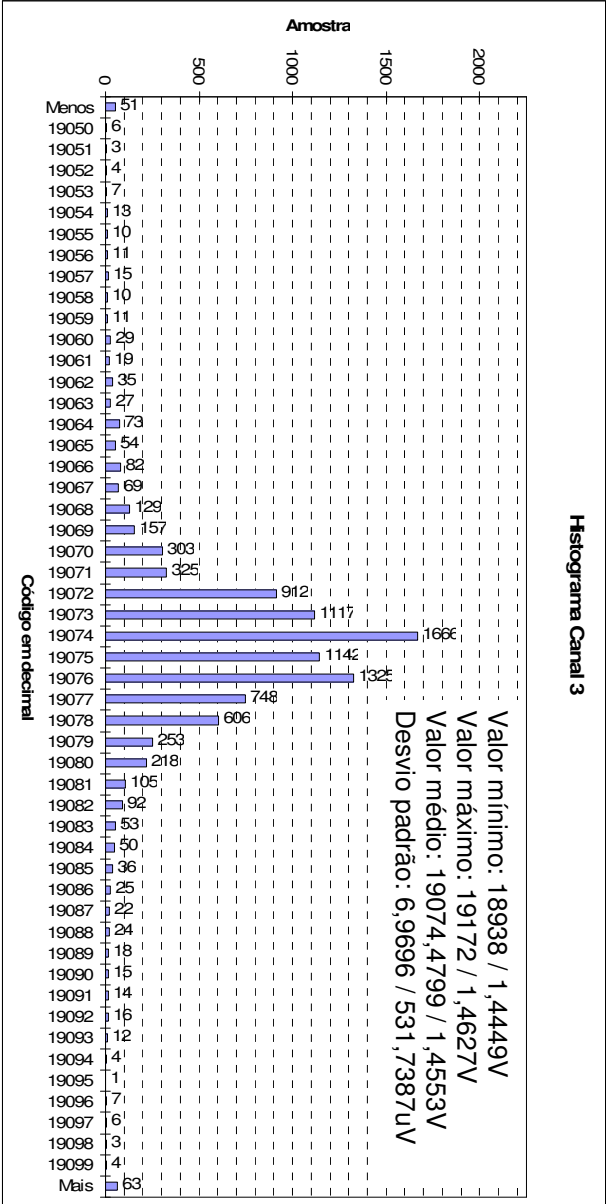


FIGURA 21 - HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 3

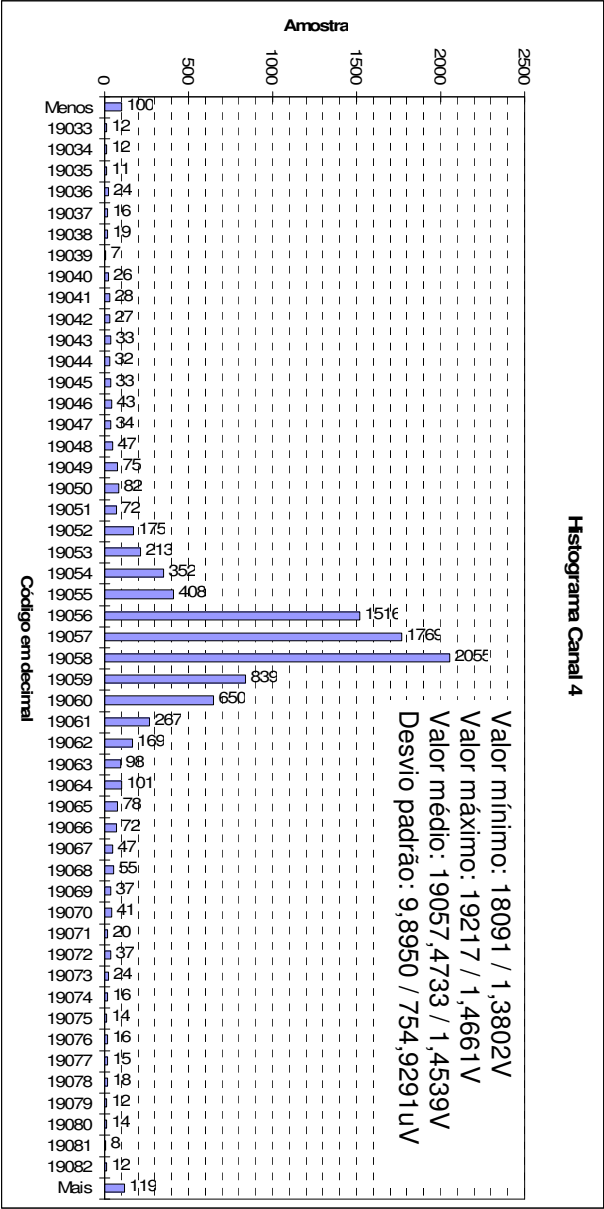


FIGURA 22 – HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 4

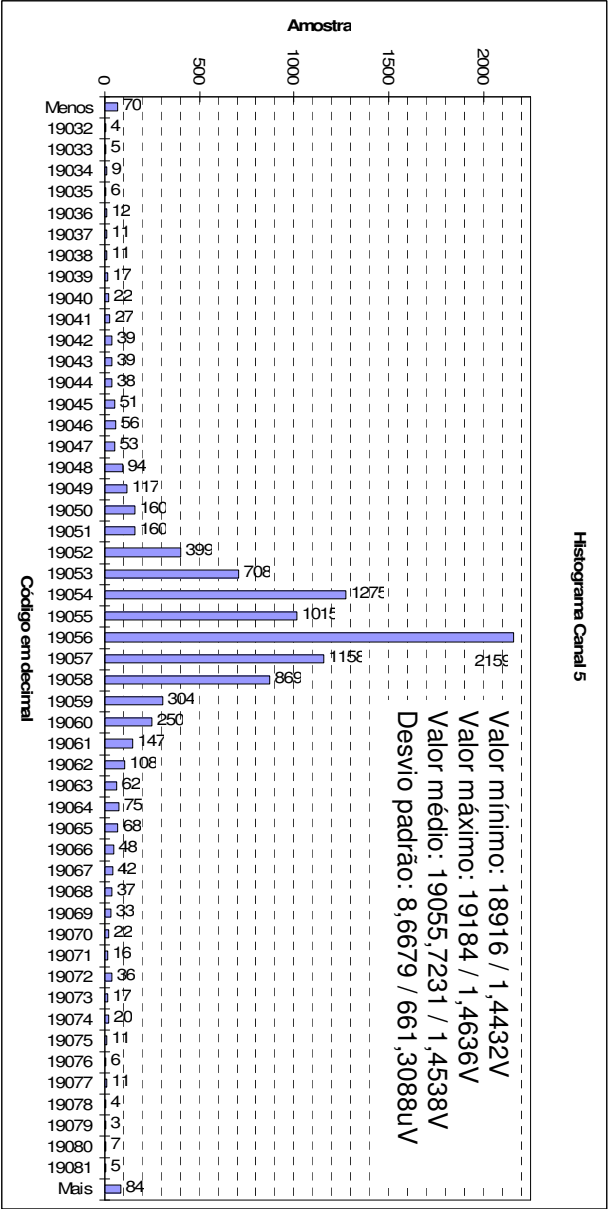


FIGURA 23 – HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 5

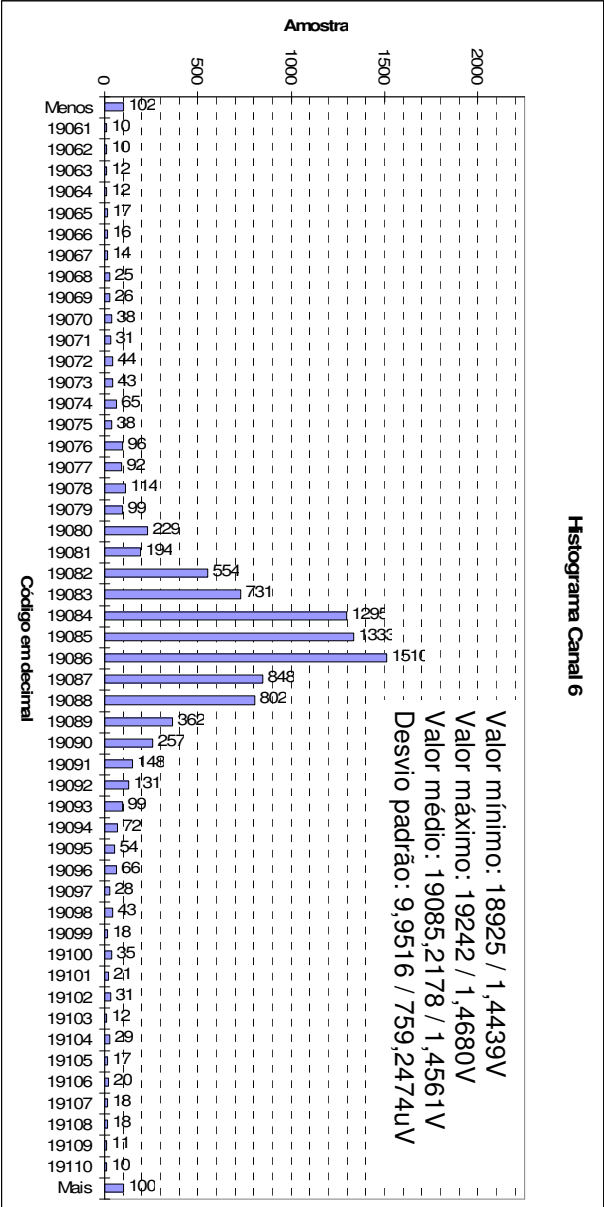


FIGURA 24 – HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 6

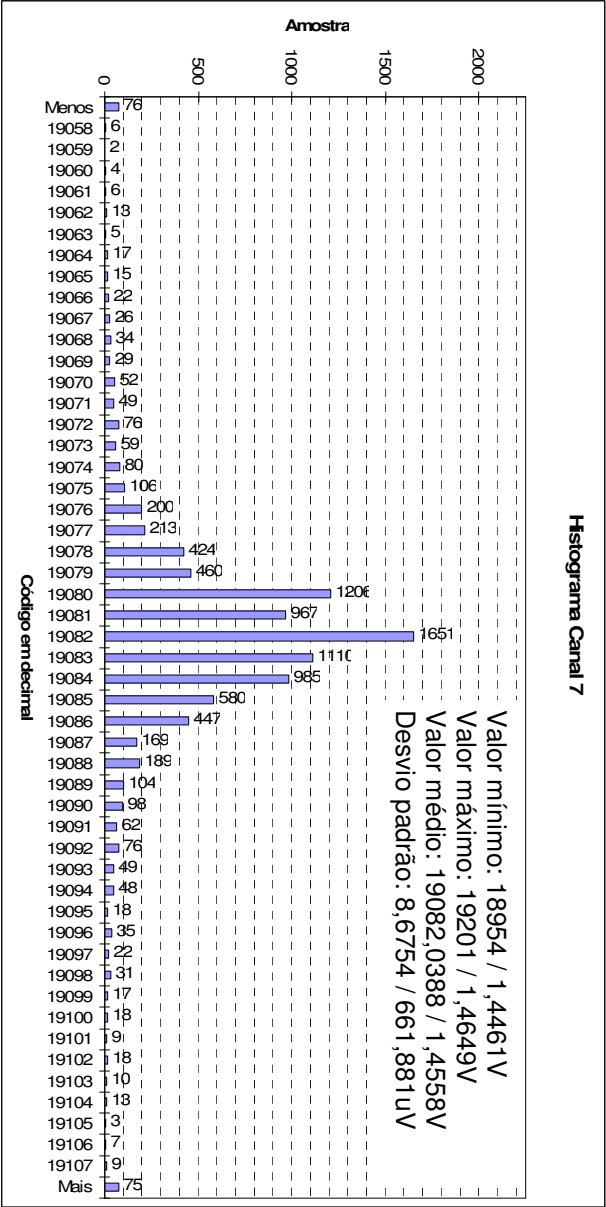


FIGURA 25 - HISTOGRAMA PARA TENSÃO CONTÍNUA NO CANAL 7

Analisando os gráficos dos canais CH_4, CH_5, CH_6 e CH_7, (serão

desconsiderados os canais os canais CH_0 a CH_3), o maior desvio padrão é 9,9516, que equivale a uma tensão inferior a 800 μ V. As médias que fazem parte de um mesmo canal, são bastante próximas, tendo uma diferença inferior a 4. Mas a diferença das médias entre canais de ADCs diferentes apresenta um erro significativo, sendo inferior a 30.

Estas diferenças podem ser atribuídas aos modelos de ADCs diferentes que foram usados (AD7654 e AD7655) e também ao *layout* da placa, pois os componentes estão dispostos de forma que as trilhas que levam o sinal para cada canal do ADC percorrem caminhos de comprimento diferente. Estes problemas poderiam ser minimizados utilizando referências de tensão distintas, sendo cada uma calibrada para atender as necessidades de cada ADC.

3.1.3 Módulo ZigBee

Um teste muito simples foi realizado para determinar o alcance de comunicação dos módulos ZigBee. Foi escrito um pequeno *software* de teste que alterna um LED constantemente, enquanto há comunicação com o módulo. O teste foi realizado sem nenhum obstáculo entre os módulos (alcance visado). A placa de recepção foi deixada fixa, disposta a uma altura de 1,5 metros do chão. Vagarosamente, a placa de aquisição foi sendo afastada até que o LED parasse de piscar. Desta maneira foi determinada uma distância de alcance superior a 100 metros. O fabricante não informa a distância mínima de alcance dos módulos, mas a especificação ZigBee define distâncias típicas de 100m a 300m. Cabe salientar, que nenhum ensaio relacionando distância e erros de recepção foram ainda realizados.

3.2 Projeto de Software

Uma das características mais importantes, para qualquer sistema de aquisição, é a

realização cadenciada da aquisição do sinal de interesse. Para tanto, é necessária uma base de tempo confiável para a correta produção de taxas de amostragem, com tempos iguais entre cada aquisição. Rotinas de *software* podem interferir nestes tempos, portanto, um estudo cuidadoso da implementação da rotina e testes confiáveis de validação devem ser realizados.

A base de tempo do sistema implementado é produzida a partir de um cristal de 8MHz. Para garantir maior estabilidade, em função da variação de temperatura ambiente, selecionou-se um cristal com baixo coeficiente de temperatura (50PPM @ 25°C, na faixa de temperatura de -20°C a +70°C). Os capacitores de carga, ligados ao circuito oscilador, tem coeficiente de temperatura NP0. A frequência do cristal é multiplicada por quatro, por um circuito PLL integrado ao microcontrolador, e depois dividida por dois. A frequência de operação interna do microcontrolador é, portanto, 16MHz, o que dá um ciclo de máquina de 62,5ns. Este intervalo de tempo discreto é a base para gerar as taxas de amostragem do sistema.

O *software* desenvolvido utiliza as interrupções dos *timers* do microcontrolador para gerar as taxas de amostragem. Ao total, foram utilizados quatro *timers* para gerar quatro taxas de amostragem independentes, na aquisição de oito canais. Cada vez que um *timer* “estoura”, uma interrupção de *hardware* acontece. Mas no atendimento desta interrupção, inúmeras instruções de *software* são executadas até que efetivamente, uma rotina implementada seja devidamente executada. É importante que o tempo de latência, entre a ocorrência da interrupção e a execução efetiva da rotina desenvolvida, seja sempre fixo, para que a taxa de amostragem seja constante. Para garantir tempos de latência fixos, as rotinas que são executadas por cada interrupção têm a mesma estrutura, acessando a mesma quantidade de variáveis e não chamam outras funções. Também foi imposto, que o *software*, antes da ocorrência de uma interrupção, fica “esperando” por ela, executando um laço que avalia uma variável que é alterada durante o atendimento à interrupção. Estes tempos de latência foram avaliados por rotinas de teste, especialmente desenvolvidas. As medições chegaram a um valor de 16 ciclos de máquina até o início da execução da primeira instrução da rotina.

Uma das funcionalidades do *software* não teve o comportamento esperado. Os *timers 2 e 4* são disparados juntos, cerca de 1000 ciclos de máquina depois, os *timers 3 e 5* são disparados. Isto implica que, quando a interrupção do *timer 2* estiver sendo atendida, a interrupção do *timer 4* pode ocorrer. Como as duas interrupções têm a mesma prioridade, a execução atual não é interrompida, e a instrução *DISI* vem a desabilitar qualquer outra interrupção que possa ocorrer. Até aqui não há problemas, principalmente se ambos os *timers* tem o mesmo valor. Comportamento estranho ocorre quando os *timers* têm valores diferentes. Por exemplo, o *timer 2* com valor de “estouro” 1000 e o *timer 4* com valor de “estouro” em 2000 contagens. Assim que há o “estouro” *timer 2*, e ele é devidamente zerado, o *timer 4* também é zerado. Este problema também vem ocorrendo nos *timers 3 e 5*. O problema ainda está sendo investigado para futura resolução.

Finalizando o processo de testes, o sistema foi avaliado em diversas taxas de amostragem entre 1 amostra por segundo e 5000 amostras por segundo, com todos os canais habilitados. Fica registrada a limitação de que, no arquivo de configuração informado pelo usuário, deve-se passar uma combinação de valores de taxas de amostragem que gerem o mesmo tempo para o *timer 2 e 4* assim como para os *timers 3 e 5*. Por exemplo, entre os canais CH_0, CH_4 e CH_2, CH_6, deve haver ao menos um valor em comum. Obviamente o *buffer* de memória informado não pode passar de 2097152 *words*.

CONCLUSÕES

Projetos de sistemas de aquisição de alta resolução necessitam de inúmeros cuidados. Ruídos muito pequenos causam erros significativos nas conversões analógico-digitais, portanto, fontes, reguladores, filtros e referências de tensão devem ser projetados com especificações rigorosas para atender os parâmetros desejados. Placas de circuito impresso de multi camadas, com planos de terra separados para os blocos digitais e analógicos são uma premissa no desenvolvimento do *layout*.

As técnicas citadas foram implementadas durante o desenvolvimento do sistema de baixo custo proposto. Sendo esta a primeira versão do projeto, algumas limitações foram observadas e, merecem investigação futura, no caso de uma segunda versão. O *layout* da placa de circuito impresso pode ser melhorado, fazendo com que os planos de terra analógico e digital unam-se mais próximos à fonte de entrada. Os ADCs podem ser melhor dispostos na placa e, referências de tensão independentes para cada um, podem melhorar o desempenho do sistema. É recomendado que os pinos dos ADCs que iniciam o processo de conversão, sejam isolados do microcontrolador, inserindo um circuito condicionador entre eles, para evitar a indução de ruídos durante a conversão. Durante os testes de avaliação do sistema, verificou-se que os canais CH_0 a CH_3 possuem uma não linearidade entre tensões de 0V a aproximadamente 10mV, problema que não ocorre nos canais CH_4 a CH_7, que operam linearmente entre toda a faixa de 0V a 5V. Mesmo assim obteve-se uma boa repetibilidade nas medidas durante os ensaios, com desvio padrão inferior a 800μV nos

canais avaliados.

Sendo o protocolo ZigBee bastante robusto para a implementação de redes *wireless*, durante os testes realizados, não foram observados discrepâncias nos pacotes de dados recebidos pela placa de recepção. A taxa de transmissão especificada no protocolo, 250kbps, acaba sofrendo uma diminuição razoável quando medida em relação somente aos dados efetivamente utilizados pelo usuário. Neste caso, a taxa cai para cerca de 20kbps, segundo dados do próprio fabricante. Devido o sistema implementado utilizar “confirmações” durante as transmissões de dados, a taxa efetiva de dados é reduzida ainda mais. Nos ensaios realizados, para transmitir um *buffer* de 20kB, foram medidos tempos de aproximadamente 8 minutos.

O *software* implementado, para gerar múltiplas taxas de amostragem foi avaliado, em conjunto com o hardware, para taxas de 1 amostra por segundo até 5000 amostras por segundo com todos os canais habilitados. Testes com todos os canais trabalhando com taxas de amostragem diferentes também foram realizados, comprovando esta funcionalidade do sistema. A capacidade do sistema de realizar a amostragem de até quatro canais simultaneamente, o torna bastante atrativo para aplicações que necessitem de tal característica. Há também a flexibilidade de transmitir os ensaios realizados pela interface RS-232 ou USB, podendo o sistema ser utilizado como uma placa de aquisição local.

REFERÊNCIAS BIBLIOGRÁFICAS

1. EMBER CORPORATION, **EmberZNet Application Developer's Guide**. Disponível em: http://www.ember.com/pdf/EM250/120-0066-000I_appDevGuide.pdf. Acessada em julho de 2007.
2. IEEE STANDARDS, **IEEE Std 802.15.4-2003**. USA: The Institute of Electrical and Electronics Engineers, 2003
3. MICROCHIP, **Microchip Stack for ZigBee™ Protocol**. Disponível em: <http://ww1.microchip.com/downloads/en/AppNotes/00965c.pdf>. Acessada em julho de 2007.
4. ZIGBEE STANDARDS ORGANIZATION, **ZigBee Specification**. CA: ZigBee Alliance Inc., 2006.
5. ANALOG DEVICES, **Grounding Data Converters and Solving Mystery of “AGND” and “DGND”**. Disponível em: <http://www.analog.com/en/content/0,2886,761%255F795%255F97529%255F0,00.html>. Acessada em julho de 2007.
6. TELEGESIS, **ETRX2 ZigBee Module Product Manual**. Disponível em: <http://www.telegeesis.com/pdf/Zigbee/TG-ETRX2-PM-01-103.pdf>. Acessada em julho de 2007.

7. TELEGESIS, **AT Commands Manual**. Disponível em:
<http://www.telegesis.com/pdf/Zigbee/TG-ETRX-R211-Commands.pdf>. Acessada em julho de 2007.
8. TELEGESIS, **TG-ETRX2DVK-TM-01-107**. Disponível em:
<http://www.telegesis.com/pdf/Zigbee/TG-ETRX2DVK-TM-01-107.pdf>. Acessada em julho de 2007.
9. ANALOG DEVICES, **AD7654**. Disponível em:
http://www.analog.com/UploadedFiles/Data_Sheets/AD7654.pdf. Acessada em julho de 2007.
10. ANALOG DEVICES, **AD7655**. Disponível em:
http://www.analog.com/UploadedFiles/Data_Sheets/AD7655.pdf. Acessada em julho de 2007.
11. ANALOG DEVICES, **AD780**. Disponível em:
http://www.analog.com/UploadedFiles/Data_Sheets/AD780.pdf. Acessada em julho de 2007.
12. MICROCHIP, **PIC24FJ128GA Family Data Sheet**. Disponível em:
<http://ww1.microchip.com/downloads/en/DeviceDoc/39747C.pdf>. Acessada em julho de 2007.
13. MICROCHIP, **MPLAB C30 C Compiler User's Guide**. Disponível em:
http://ww1.microchip.com/downloads/en/DeviceDoc/C30_Users_Guide_51284f.pdf.
Acessada em julho de 2007.
14. MICROCHIP, **MPLAB ASM30, MPLAB LINK30 and Utilities User's Guide**. Disponível em:
http://ww1.microchip.com/downloads/en/DeviceDoc/Asm30_Link_Util_51317f.pdf.
Acessada em julho de 2007.
15. LINEAR TECHNOLOGY, **LT1763 Series**. Disponível em:
<http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1010,C1764,P1778,D3903>.
Acessada em julho de 2007.

16. LINEAR TECHNOLOGY, **LT1963A Series**. Disponível em: <http://www.linear.com/pc/downloadDocument.do?navId=H0,C3,P2222,D3148>. Acessada em julho de 2007.
17. LINEAR TECHNOLOGY, **LT1083, LT1084, LT1085**. Disponível em: <http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1003,C1040,C1055,P1283,D3741>. Acessada em julho de 2007.
18. BALBINOT, Alexandre e BRUSAMERELLO, Valner João. **Instrumentação e Fundamentos de Medidas – Vol 1**. Rio de Janeiro: LTC, 2006.
19. SEDRA, Adel S. e SMITH, Kenneth C. **Microeletrônica**. São Paulo: Makron Books, 1999.
20. IRWIN, David J. **Análise de Circuitos em Engenharia**. São Paulo: Makron Books, 2000.
21. SCHILDT, Herbert. **Borland C++ Builder – Referência Completa**. São Paulo: Campus, 2001.
22. HEILE, Bob. **Wireless Sensors and Control Networks**. Disponível em: <http://www.zigbee.org/imwp/download.asp?ContentID=10951>. Acessada em julho de 2007.

ANEXOS