Thiago Martins de Sousa

Melhorias na plataforma de experimentos remotos VISIR Open Labs

São José – Santa Catarina

Março / 2011

Thiago Martins de Sousa

Melhorias na plataforma de experimentos remotos VISIR Open Labs

Monografia apresentada à Coordenação do Curso Superior de Tecnologia em Sistemas de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do diploma de Tecnólogo em Sistemas de Telecomunicações.

Orientador:
Prof. Ingvar Gustvasson, Dr. Eng.

Co-orientador:
Prof. Eraldo Silveira e Silva, M. Eng.

CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES
INSTITUTO FEDERAL DE SANTA CATARINA

São José – Santa Catarina

Março / 2011

Monografia com o título de *Melhorias na plataforma de experimentos remotos VISIR Open Labs*, apresentada por Thiago Martins de Sousa em 31 de Março de 2011 , em São José, Santa Catarina, pela banca assim constituída:

Prof. Eraldo Silveira e Silva, M. Eng.
IFSC

Prof. Evandro Cantú,Dr. Eng.
IFSC

Prof. André Luiz Alves, Eng.
IFSC

”O único lugar onde o
sucesso vem antes do
trabalho é no dicionário.”
Albert Einstein

Agradecimentos

Eu gostaria de agradecer a Deus por me permitir nascer em uma grande família. Eu gostaria de agradecer a minha família pelo apoio e por me ensinar como um homem honrado deve ser. Gostaria de agradecer ao Instituto Federal de Santa Catarina por me dar a oportunidade de um intercâmbio. Gostaria de agradecer a todos os professores do Instituto Federal de Santa Catarina pelo conhecimento que me ensinaram. Agradecer ao *Blekinge Institute of Technology* e ao professor Ingvar Gustavsson por terem me aceitado para trabalhar nesse projeto. Eu gostaria também de agradecer ao engenheiro Johan Zackrisson pela ajuda no desenvolvimento do projeto e ao escritório internacional do *Blekinge Institute of Technology* por terem me ajudado em tudo que puderam para que eu tivesse uma excelente estadia.

Resumo

Este trabalho é uma melhoria das características da plataforma de experimentos remotos *VISIR Open Labs*. Esta plataforma é um laboratório on-line para experimentos físicos de engenharia elétrica. Com VISIR os alunos acessam um site e executam experimentos preparados por um professor ou seu auxiliar. O circuito criado pelos alunos no site é montado usando placas com componentes eletrônicos e matrizes. Melhorias na comunicação entre essas placas e o suporte a conversores Analógico Digital (AD) e Digital Analógico (DA) são descritos nessa monografia. Um protocolo para a comunicação entre as placas e o computador foi criado. Seu objetivo é informar o status das placas para o computador que controla a plataforma e receber informações sobre os comandos executados, como por exemplo, erro de temporização. A compatibilidade com versões antigas foi mantida.

Palavras-Chave

Laboratórios Remotos, Protocolos de comunicação

Abstract

This project is a improvement of the features of VISIR Open Labs platform. It is a on-line laboratory for physical experiments of electrical engineering. With VISIR, students access a web site and perform experiments prepared by a staff or a teacher. The circuit performed by the students at the web site is wired using boards with electronic components and matrixes. Improvements on the communication between these boards and the support of AD/DA converter are described at this report. A new protocol for the communication between the boards and the computer was created. It purpose is to report the status of the boards to the computer which controls the platform, to read and right on a AD/DA IC converter and to receive status information about commands performed,for example timing errors. The backwards compatibility was maintained.

Keywords

Remote Labs, Protocol of Communication

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Experimentos de laboratórios são partes importante nos estudos de engenharia. Nas últimas décadas o número de estudantes cresceu, mas os investimentos em laboratórios não (Tor A. Fjeldly, 2005). Para permitir que os alunos possam executar experimentos reais vinte e quatro horas por dia e sete dias por semana os laboratórios remotos foram criados no *Blekinge Institute of Technology* (BTH). A plataforma de laboratórios remotos *VISIR Open Labs* é um desses laboratórios. Ele foi criado pelo Departamento de Processamento de Sinais (ASB), para graduandos em engenharia elétrica executarem experimentos na área de eletrônica. O software da plataforma esta disponível em http://svn.openlabs.bth.se/trac sob a licença GPL, exceto o firmware dos controladores das placas. Instituições de ensino podem se sentir encorajadas para usarem a plataforma de experimentos remotos *VISIR Open Labs* para abrir seus próprios laboratórios remotos e participar de suas futuras pesquisas e desenvolvimento.

Para acessar a plataforma de experimentos remotos *VISIR Open Labs* os estudantes de engenharia elétrica do BTH têm que acessar a página da plataforma[1] e efetuar *login*. Após feito o *login*, os estudantes selecionam os experimentos preparados anteriormente por um professor ou seu auxiliar. O experimento selecionado é mostrado em uma página escrita em *Flash*. Nesta página os estudantes tem disponível uma matriz de contato, componentes eletrônicos pré selecionados como resistores e capacitores, e equipamentos de bancada como osciloscópio e fontes de tensão. Para dar impressão de realidade os painéis dos equipamentos de bancadas são imagens fiéis dos painéis dos equipamentos disponíveis no laboratório. Os estudantes selecionam os componentes a serem usados, conectam-nos na matriz de contato e interligam os componentes. Após isto, devem ser selecionados os instrumentos de bancadas a serem usados e interligá-los com os componentes. Os instrumentos disponíveis são fonte de tensão, gerador de funções, multímetro digital e osciloscópio. Após finalizar as conexões com os equipamentos de bancada, o estudante clica no botão para executar o experimento. A plataforma de experimentos remotos *VISIR Open Labs* monta o circuito criado pelo estudante usando uma matriz feita de placas com relés. As tensões e correntes são geradas e medidas por equipamentos

[1] http://openlabs.bth.se/electronics

reais que estão em um laboratório do BTH. As placas que comutam os relés para conectar os componentes e os equipamentos foram desenvolvidas pelo BTH. O trabalho que consta nesse documento é sobre as melhorias feitas na comunicação entre essas placas da matriz e também com o computador que as controla.

1.1 Motivações do trabalho

A versão atual da plataforma citada possui duas grandes limitações. Uma é o fato de que os experimentos suportados são limitados a componentes analógicos. A versão atual do firmware usado nos micro controladores não suporta experimentos com componentes digitais ou analógicos e digitais como CI's conversores AD/DA, mas um barramento para esta função já existe. Os experimentos citados são comuns e necessários para o estudo da engenharia elétrica. Outra limitação da plataforma é o fato que a comunicação entre as placas da matriz de relés usada para montar o circuito bem como a comunicação entre a comunicação entre a matriz e o computador é somente em um sentido(GUSTAVSSON; ZACKRISSON, 2007). Não há confirmação da execução do comando ou alguma informação sobre o *status* das placas. Isto dificulta o gerenciamento da plataforma e pode camuflar erros. Os objetivos deste trabalho são:

- Implementar o suporte a comunicação em dois sentidos entre o *Equipment Server* e a matriz.
- Implementar o suporte ao uso do barramento digital para controlar CI's conversores AD/DA .
- Criar os *drivers* em *LabView* para as novas funcionalidades.

Para alcançar os objetivos citado um protocolo de comunicação simples foi criado.

1.2 Organização do texto

Na introdução foi mostrado a motivação e os objetivos a serem alcançados por este trabalho. Para melhor organização do texto a descrição completa da plataforma já existente, tanto do hardware como do software é feita no capítulo 2. O novo protocolo de comunicação entre as placas e entre o computador é feita no capítulo 3. No capítulo 4 são mostrados exemplos de uso e comparações com a versão atual. As conclusões e trabalhos futuros são mostrados no capitulo 5.

2 VISIR Open Labs

2.1 Visão geral

Na plataforma de experimentos remotos *VISIR Open Labs* os equipamentos de bancada de um laboratório comum foram substituídos por um único equipamento. O equipamento usado é um chassi NI PXI-1033 [2] produzido pela *National Instruments*. Este equipamento é modular e possui um módulo para realizar a função de cada equipamento de bancada. O chassi usado na plataforma possui um módulo para multímetro digital, gerador de funções, osciloscópio e fonte de tensão. As saídas e entradas destes módulos são conectadas à matriz de relés que é uma pilha de placas. Este equipamento é controlado por um computador. A figura 2.1 mostra a matriz, o chassi com os equipamentos de medição e o computador que o controla.

Figura 2.1: Matriz e NI PXI-1033 (GUSTAVSSON; ZACKRISSON, 2007)

A Matriz mostrada na figura 2.1 substitui a matriz de contato e os seus relés substituem as conexões feitas com fios nos circuitos. Esta matriz foi desenvolvida pelo ASB para conectar os componentes entre si e conectar aos equipamentos de medição do chassi NI PXI. As possibilidades de disposição dos componentes no circuito e limites da tensão gerada pela fonte de

[2] http://www.ni.com/pxi/

tensão e gerador de funções são configuradas anteriormente pelo professor ou seu auxiliar que montaram o experimento.

A plataforma é divida em cinco partes.

- *Web Server*.
- *Measurement Server*.
- *Equipment Server*.
- *Experiment Client*.
- *Database*.

O diagrama de blocos da plataforma de experimentos remotos *VISIR Open Labs* é mostrado na figura 2.2.

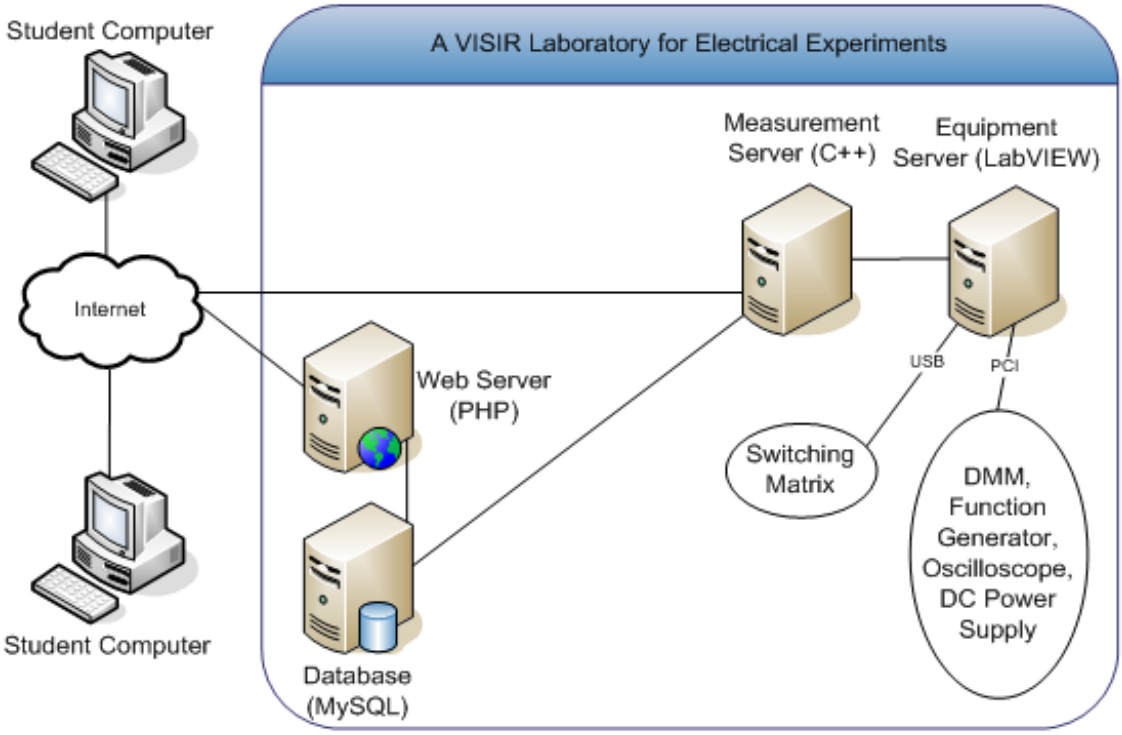

Figura 2.2: Diagrama de blocos da plataforma *VISIR Open Labs* (GUSTAVSSON, 2008)

2.1.1 *Web Server*

Servidor Web *Apache* com páginas *Hypertext Markup Language* (HTML) que usam *PHP* para a interação com usuário. Este servidor é responsável pela administração do sistema, ad-

missão de usuário e agendamento de recursos. Através dele o estudante carrega o *Experiment Client*. Os dados sobre a administração do sistema que ele utiliza estão contidos na *Database*.

2.1.2 *Measurement Server*

Este servidor tem a função de validar as requisições feitas pelo *Experiment Client*. Na requisição é enviado o circuito criado pelo estudante, esta informação esta contida em um arquivo *Extensible Markup Language* (XML). Esta informação é traduzida e validada. Se o circuito montado pelo aluno não é válido ele retorna uma mensagem de erro ao *Experiment Client*. Se for válido ele envia a requisição para o *Equipment Server* montar o circuito e realizar as medições e espera a resposta. Após receber as medições o *Measurement Server* envia os resultados obtidos para o *Experiment Client*.

2.1.3 *Equipment Server*

Servidor que controla o chassi NI PXI e a matriz. Este servidor foi escrito usando o *software LabView*. O *Equipment Server* comunica com o chassi NI PXI usando barramento *Peripheral Component Interconnect* (PCI) e com a matriz usando o barramento *Universal Serial Bus* (USB).

2.1.4 *Experiment Client*

Módulo escrito em *Adobe Flash* que roda na página web acessada pelo cliente. Ele é responsável pela interação com o estudante. É nele que é mostrado aos estudantes a matriz de contato virtual, os componentes disponíveis e os equipamentos de bancada. Usando o mouse os estudantes podem selecionar componentes, conectá-los à matriz de contato virtual, conectar os componentes através das trilhas da matriz de contato ou com fios, bem como os equipamentos de bancada. Para dar à impressão de realidade a matriz de contato, os componentes e os equipamentos de bancada são fotos de equipamentos reais como é mostrado na figura 2.3.

Após montar o circuito e o estudante executar o experimento o *Experiment Client* envia a disposição dos componentes no circuito, equipamentos de bancada selecionados e as tensão e correntes de entrada no circuito para o *Measurement Server* validá-lo e prosseguir a execução do experimento.

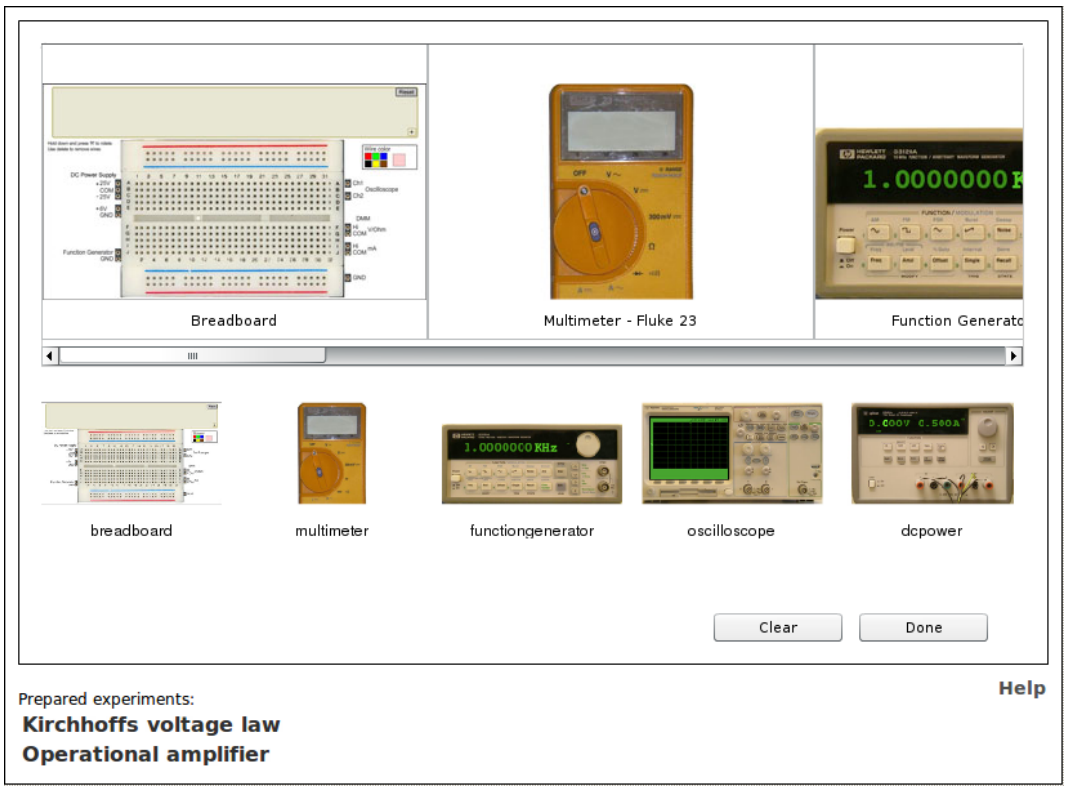

Figura 2.3: Tela do *Experiment Client*

2.1.5 *Database*

A *Database* é à base de dados do sistema. É nela que o *Web Server* faz as pesquisas de dados referentes ao gerenciamento da plataforma como *login*, agendamento de recursos, números de experimentos. O *Measurement Server* também o utiliza para fazer consultas sobre a lista de componentes e limites de tensões e correntes de entrada.

2.1.6 Exemplo de utilização

Para acessar os experimentos os estudantes precisam efetuar o *login* na pagina hospedada no *Web Server*. O *login* envolve o *Web Server* e o *Database*. Ao escolher o experimento a ser executado, uma página *HTML* contendo o *Experiment Client* é carregada para no navegador do estudante. No *Experiment Client* o estudante pode selecionar os instrumentos de bancadas que serão usados e a disposição dos componentes no circuito. Após selecionar os componentes disponíveis, montar o circuito com o mouse e mandar executá-lo o *Experiment Client* envia requisição ao *Measurement Server* para validar o circuito. Esta requisição esta no formato de um arquivo XML. Este arquivo contem *tags* para descrever quais equipamentos de bancadas foram selecionados, componentes usados, disposição dos componentes e suas

conexões com os equipamentos. A descrição completa do arquivo XML está disponível em http://svn.openlabs.bth.se/trac/measureserver/wiki/ClientProtocol. O *Measurement Server* faz a validação baseado na disposição dos componentes conectados à matriz, limites de máximos de tensão e corrente aceitos pelos componentes, e também pelos limites máximos de tensão e corrente gerado pela fonte de tensão ou gerador de funções. Essas informações são de responsabilidade do professor responsável pelo experimento ou seu auxiliar. Essa validação é feita para evitar danos aos componentes ou equipamentos usado no experimento. Se a requisição é validada o *Measurement Server* envia o circuito montado pelo aluno para o *Equipment Server*. Este então envia os comandos para a matriz comutar os relés e conectar os componentes entre si e com o NI PXI e também para setar os valores de tensão e correntes usadas pela fonte e gerador de funções. O *Equipment Server* também lê as medidas feitas pelo *Measurement Server* e as envia para o *Experiment Client* que mostra os resultados do experimento no *display* dos seus equipamentos de medição como mostrado na figura 2.4

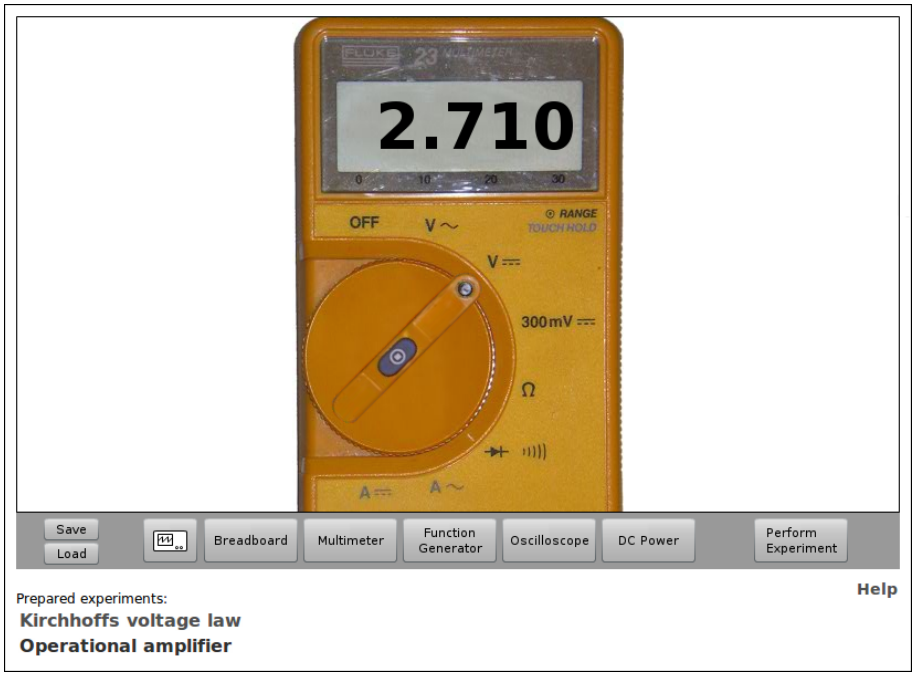

Figura 2.4: Resultado da medição na tela do multímetro

2.2 Hardware da Matriz

Uma descrição mais aprofundada do hardware da matriz que simula a matriz de contato é feita, pois a maior parte do trabalho realizado pelo aluno e descrito no capítulo 3 foi escrever o *firmware* dos micro controladores presentes nestas placas.

Um exemplo de uma matriz com quatro placas usadas na plataforma de experimentos re-

motos *VISIR Open Labs* é mostrada na figura 2.5.

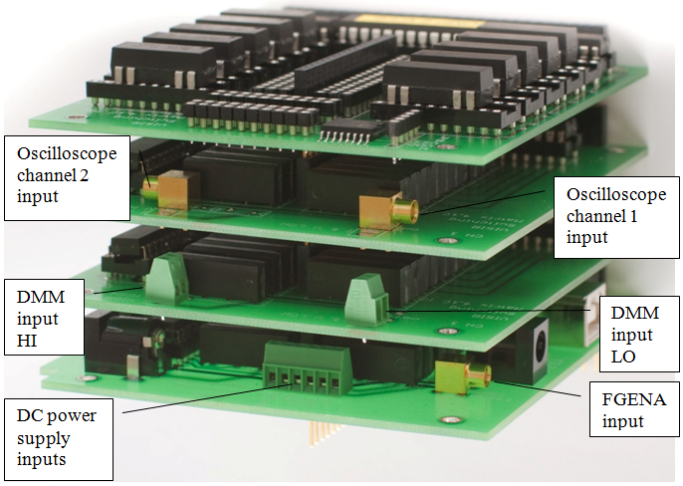

Figura 2.5: Matriz de relés (GUSTAVSSON, 2008)

Todas as placas da matriz seguem o padrão PC/104[4]. Este é um padrão de construção de placas para computação embarcada. Ele define a pinagem e a dimensão das placas. Cada placa usada possui dois conectores, isto permite que várias placas sejam empilhadas e compartilhem o mesmo barramento, eliminando a necessidade de backplanes ou bastidores. Os conectores são chamados de :

- Conector de nós
- Conector do barramento digital

O Conector de nós possui 17 pinos e cada pino é chamado de nó. Cada um destes 17 pinos poderá ser usado como um ponto para conexão entre componentes e equipamentos, ou seja, um nó do circuito montado. Os nós são denotados de A até I, X1 até X6, 0 e COM. O nó 0 é conectado ao GND da placa.

O Conector do barramento digital possui 28 pinos. Ele é ligado ao micro controlador que controla a pilha de placas. Apenas quatro pinos são usados na versão atual da plataforma, dois são usados como barramentos de comunicação entre os micro controladores das placas, um para prover +5V e outro para GND. Mais pinos serão usados na nova versão do software descrita no capítulo 3.

Além dos dois conectores citados acima, todas as placas possuem um micro controlador PIC16F767 e relés. O PIC16F767 é chamado de *Board Controller*, ele controla os relés. A

[4]http://www.pc104.org

alteração do estado dos relés para fechado ou aberto é feita quando o *Board Controller* recebe um comando pelo barramento digital. Na figura 2.5 pode ser notado que as placas da matriz não são iguais. Existem três tipos distintos de placas, elas são:

- *Component Board.*
- *Instrument Board.*
- *Source Board.*

2.2.1 *Component Board*

Esta placa é a mais comum da matriz. Ela é usada para conectar componentes aos nós do circuito ou para conectar a circuitos externos à matriz. Os componentes eletrônicos usados nos experimentos são conectados a ela. Uma *Component Board* é mostrada na figura 2.6.

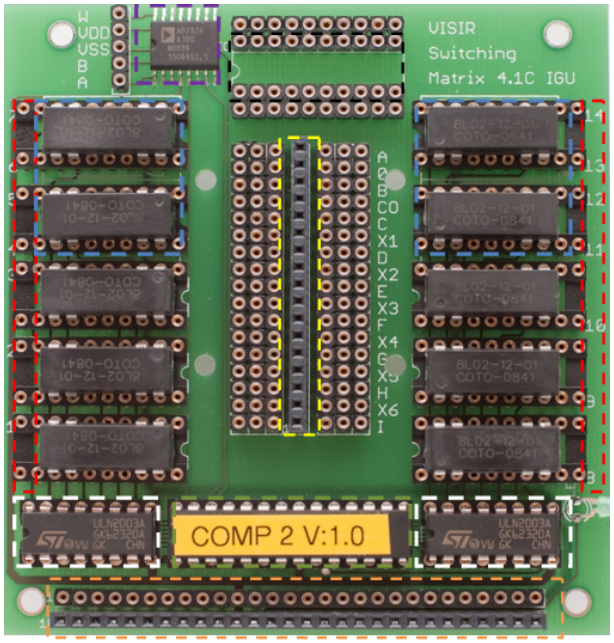

Figura 2.6: *Component Board*

Nela pode ser notado que algumas partes da figura estão pontilhadas para destacar detalhes da placa. A área pontilhada em amarelo é o conector de nós. Os conectores fêmeas ao lado dele são usado para conectar os relés ao barramento. A área pontilhada em laranja é o conector do barramento digital. A área pontilhada em vermelho mostra os conectores para componentes de duas pernas como resistores e capacitores. Esses conectores são numerados de 1 a 14. Os quatro relés de dois pólos na área pontilhada em azul podem ser substituídos por oito relés de um pólo para dar maior flexibilidade na construção do circuito, por isso essa é a configuração padrão. O

conector de 20 pinos na área pontilhada em preto é reservado para componentes ou CI's com mais de duas pernas, ele não tem conexão física com o resto do circuito da placa. Para usá-lo é necessário conectar fios entre ele e o resto do circuito. O CI na área pontilhada em roxo é um potenciômetro digital, modelo AD7376 que é usado em experimentos com resistência variável. O CI na área pontilhada em verde é o *Board Controller*. Os CI's ULN2003A na área pontilhada em branco são usados adequar

as características elétricas da saída do *Board Controller* com a entrada dos relés. Um diagrama de blocos da placa é mostrado na figura 2.7.

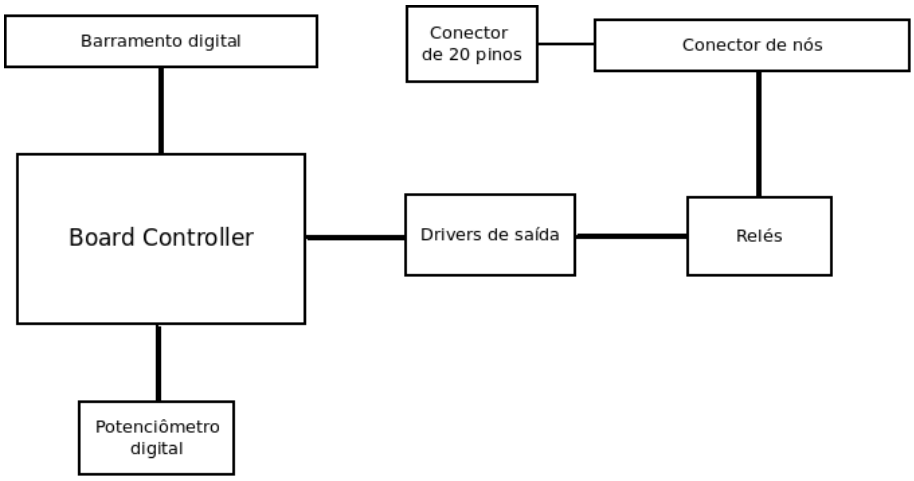

Figura 2.7: Diagrama de blocos da *Component Board*

Um exemplo de como um resistor pode ser conectado aos nodos A e B de uma *Component Board* é mostrado na figura 2.8.

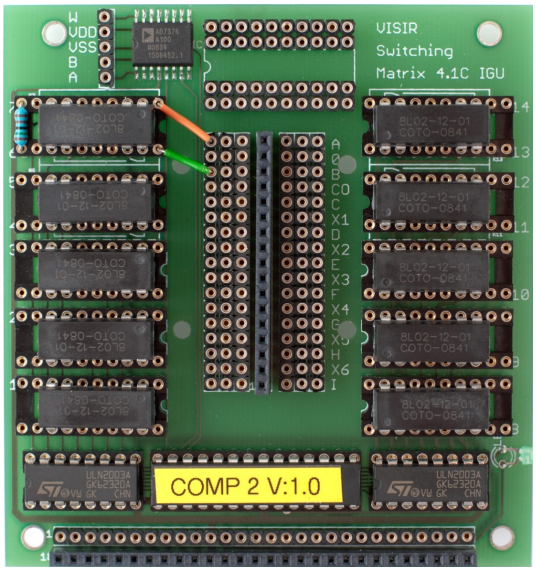

Figura 2.8: *Component Board* com um relé conectado aos nodos A e B (GUSTAVSSON, 2008)

Na figura 2.8 o resistor está conectado aos relés 6 e 7. Estes relés estão conectados ao nodo A e B do barramento de nós através de fios de cor verde e laranja. Com estas conexões feitas, para que o resistor fosse conectado aos nodos A e B seria necessário que os relés na posição 6 e 7 fossem fechados.

2.2.2 *Instrument Board*

Placa usada para conectar o chassi NI PXI aos nós do circuito montado para aplicar tensões ou para efetuar a medição. Na figura 2.5 a segunda e terceira placa são do tipo *Instrument Board*. Elas possuem diferentes conectores, a segunda placa de baixo para cima possui conectores para o multímetro digital enquanto a terceira possui conectores para o osciloscópio.

2.2.3 *Source Board*

Placa controladora da matriz. Ela controla todos os *Board Controllers* das outras placas da matriz, por isto em cada matriz só poderá haver uma *Source Board*. Ela possui um micro controlador a mais, que é um PIC18F4550 (MICROCHIP, b), chamado de *Matrix Controller*. O *Matrix Controller* possui uma interface USB que é conectado ao *Instrument Server* para receber os comandos a serem executados. Ele é também o mestre do barramento I2C usado para a comunicação entre os Controladores da placa. Esta placa possui conectores para a fonte de tensão e o gerador de funções do chassi NI PXI. A placa *Source Board* possui também conectores auxiliares para entradas de +6/+20/-20 volts.

2.3 Comunicação entre as placas

A matriz comandada pelo *Instrument Server*, que envia os comandos para o *Matrix Controller* que reenvia para as placas. Para isto é usado o protocolo I2C e USB. A figura 2.9 mostra o *Data Path* da comunicação.

2.3.1 USB

USB (Universal Serial Bus) é uma especificação para estabelecimento de comunicação entre um dispositivo e seu controlador. Quatro tipos de transferência de dados são definidos pela especificação do USB. Na plataforma de experimentos *VISIR Open Labs* é usado o *Bulk Transfer*(USB.ORG,), pois ele faz a checagem e correção de erros garantindo a entrega das

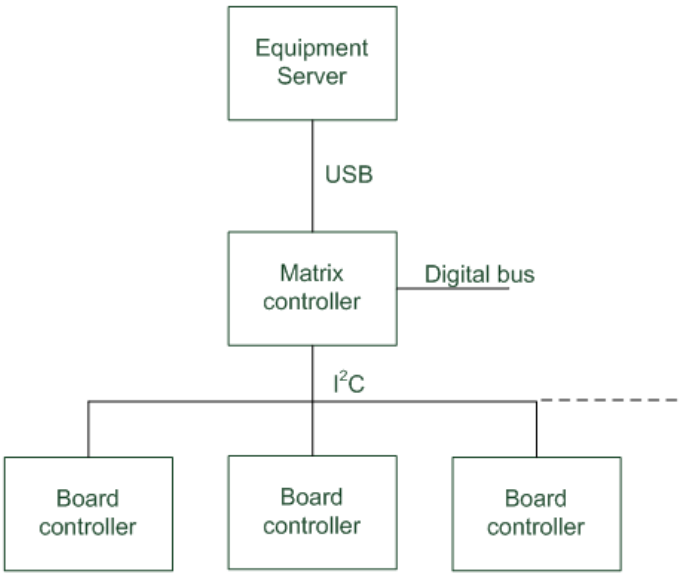

Figura 2.9: *VISIR Open Labs data path*

informações.

2.3.2 I2C

O I2C é um barramento serial do tipo mestre escravo criado para conectar periféricos. Ele utiliza dois pinos, um que transmite o *clock* e é chamado de *Serial Clock* (SCL) e outro que transmite os dados e é chamado de *Serial Data Line* (SDA) (NXP,). Na matriz o barramento I2C usa dois pinos do conector do barramento digital. Estes pinos estão conectados a todos os *Board Controllers* e ao *Matrix Controller* que é o mestre do barramento. Somente o mestre pode iniciar uma transferência de dados. Para poder acessar os dispositivos do barramento o protocolo I2C define um esquema de endereços. São endereços de 8 bits onde, onde os 7 bits mais significativos são o endereço no barramento I2C e o bit menos significativo indica se o mestre está escrevendo ou lendo o escravo. Na matriz usada no *Visir Open Labs*, este endereço é gravado no *firmware* do *Board Controller*, para identificar o endereço um adesivo é colado em cima do *Board Controller*. Na Matriz alguns endereços I2C são reservados para placas específicas como mostrados na tabela 2.1. Os endereços não citados na tabela 2.1 estão disponíveis para as outras placas.

2.4 Versão atual do protocolo

Na versão atual do protocolo, ele é dividido em duas partes. Uma é a comunicação entre o *Equipment Server* e o controlador da matriz que é chamado de *Matrix Protocol* e a outra é entre

Endereço I2C	Tipo de placa	Adesivo
1	*Component Board* 1	COMP 1
2	*Component Board* 2	COMP 2
16	*Instrument Board* (conector para multímetro)	OSC 16
17	*Instrument Board* (conector para osciloscópio)	DMM 17
24	*Source Board*	SRC 24

Tabela 2.1: Esquema de endereços I2C

o *Matrix Controller* e os controladores das placas que é chamador de *Board Controller*. Eles são descritos abaixo.

2.4.1 *Matrix Protocol*

Quando o status de um relé precisa ser alterado o *Equipment Server* envia uma mensagem para o *Matrix Controller*s pela *USB* em formato *RAW*. Todos os comandos enviados pelo *Equipment Server* para o *Matrix Controller* através da USB possuem 64 bytes, mas somente os primeiros 12 bytes são usados. Isto ocorre porque na USB esta sendo usado transferência do tipo *Bulk Transfer* que usa quadros de 64 bytes para efetuar a transmissão. Nesta mensagem é enviado o endereço da placa alvo e o novo estado dos relés. Estes dados estão em 4 bytes que são codificados em 12 bytes para serem enviados. Na informação original de 4 bytes o primeiro byte é o endereço da placa e os outros bytes são o estado dos relés. O ultimo byte pode ser também a configuração da resistência do potenciômetro digital caso a placa alvo for uma *Component Board*. A informação original com 4 bytes e sua correspondência entre bytes e relés é mostrada na tabela 2.2.

Byte 0	Byte 1	Byte 2	Byte 3
Endereço I2C	Relé 1 – 7	Relé 8 - 14	Relé 15 - 21 ou conf. potenciômetro

Tabela 2.2: Informação original e correspondência entre bits e relés

Antes da informação original citada na tabela 2.2 ser enviada o valor de cada byte é mapeado em três caracterees representando a centena, dezena e unidade em decimal do valor de cada byte. Cada caractere é codificado em ASCII conforme seu valor da tabela ASCII gerando uma mensagem de 12 bytes. Por exemplo, se o byte 1 tiver valor 24 em decimal ele ira gerar três caracteres que serão 0, 2, 4. Cada um destes caracteres será codificado em ASCII gerando os bytes que teram os valores 0x30, 0x32 e 0x34. Estes valores na tabela ASCII são respectiva-

mente 0, 2, 4. Isso se aplicará para todos os quatro bytes da informação original. A figura 2.10 mostra o formato da nova mensagem após a codificação da tabela ASCII.

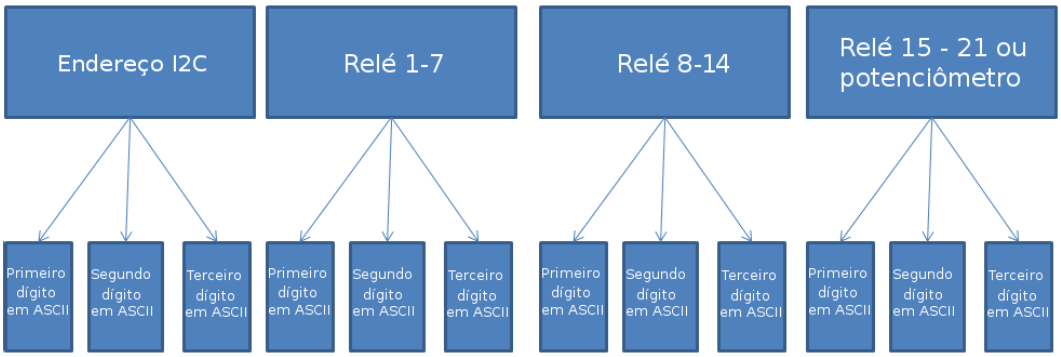

Figura 2.10: Formato do comando *Matrix Protocol* após codificar em ASCII

Após codificar a informação original conforme a tabela ASCII o *Equipment Server* envia o comando para o *Matrix Controller*. Após receber o comando o *Matrix Controller* reenvia essas informações para o *Board Controller* da placa alvo usando o *Board Protocol*. O *Matrix Controller* não envia nenhuma resposta sobre a execução do comando para o *Equipment Server*.

2.4.2 *Board Protocol*

Este protocolo é usado para enviar comandos para as placas usando o barramento I2C. O *Matrix Controller* decodifica os 12 bytes recebidos em quatro bytes. O primeiro byte é o endereço I2C da placa alvo e os outros três bytes são informações sobre os relés a serem alterados. O último byte pode ser também a configuração da resistência do potenciômetro se a placa alvo é uma *Component Board*. O formato do comando é mostrado na tabela 2.3.

Tipo de placa	Byte 0	Byte 1	Byte 2	Byte 3
Source	Endereço I2C da placa	relé 1 – 7	relé 8 - 14	Não usado
Instrument	Endereço I2C da placa	relé 1 – 7	relé 8 - 14	relé 15 - 21
Component	Endereço I2C da placa	relé 1 – 7	relé 8 - 14	Conf. do potenciômetro

Tabela 2.3: *Board Protocol*

2.4.3 Exemplo do uso

Um comando para conectar um resistor a dois nós do barramento de nós é um simples exemplo do protocolo existente. Na figura 2.8 é mostrado uma *Component Board* tem um resistor conectado ao relé 6 e 7. Caso a placa tenha o endereço I2C com valor 2 e se deseja conectar o resistor aos nós A e B o *Equipment Server* iria atribuir o valor 2 ao primeiro byte, o valor 96 ao segundo byte e os outros dois bytes teriam valor zero. Gerando assim uma informação inicial de quatro bytes com valores de 2, 96, 0, 0 .O primeiro byte teria valor 2 pois seria o endereço I2C da placa, o segundo byte teria valor 96 pois os bits b5 e b6 que são os dois bits mais significativos entre os 7 bits que carregam informação dos relés estariam setados conforme mostra a tabela 2.4.

Valor do byte	Bits correspondentes aos relés
96	1100000

Tabela 2.4: Exemplo do mapeamento de bits correspondente aos relés em um byte

O valor de cada um dos quatro bytes gerados seria mapeado em três caracteres. Cada caractere seria codificado seguindo o seu valor da tabela ASCII. Isto iria gerar 12 bytes. A tabela 2.5 mostra o comando em ASCII que seria enviado.

Comando original	2			96			0			0		
Correspondente em caracteres	0	0	2	0	9	6	0	0	0	0	0	0
Comando com caracteres codificados em ASCII enviado	0x30	0x30	0x32	0x30	0x39	0x36	0x30	0x30	0x30	0x30	0x30	0x30

Tabela 2.5: Comando com caracteres codificados em ASCII

O comando enviado para o *Matrix Controller* teria os byte 0x30, 0x30, 0x32, 0x30, 0x39, 0x36, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30.

A figura 2.11 mostra o driver escrito em *LabView* que faz a conversão a codificação e envio desse comando.

Na figura 2.11 o bloco *VISA resource name* é a identificação do dispositivo que o *Equipment Server* irá fazer o envio do comando. Neste caso é a matriz de placas. O bloco em azul com o nome *Arrays of board numbers and bit mask* é o responsável por montar a informação de 4 bytes com o primeiro byte sendo o endereço I2C da placa e os próximos três bytes a mascara de bits do estado dos relés. O bloco *Test without hardware* é usado para testar somente a conversão dos 4 bytes em um comando de 12 bytes sem fazer o envio para a placa. O bloco *error in(no error)* é um código de erro com valor zero definido pelo *Labview*. Após a transmissão este

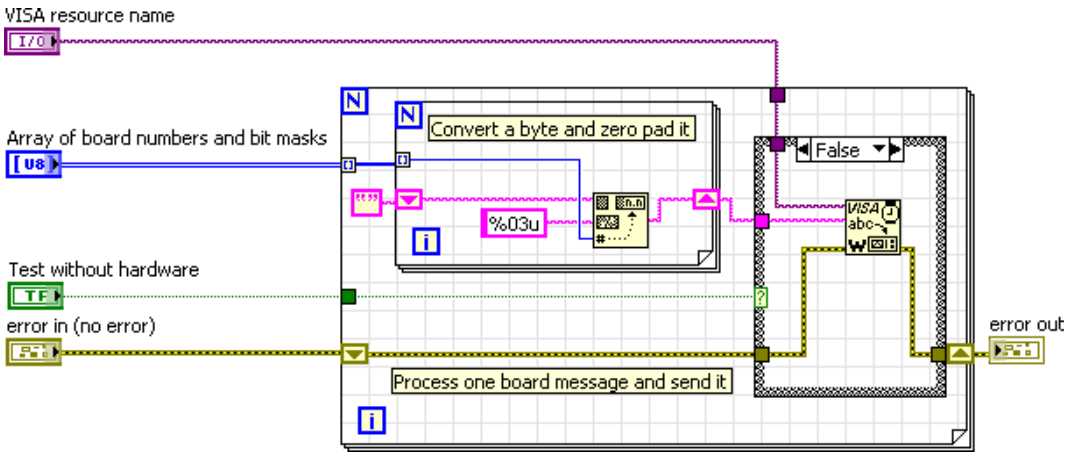

Figura 2.11: Driver simples em *LabView* para conversão e envio de comando

código é comparado com o *error out* para verificar se a interface USB identificou algum erro. No centro da imagem há uma estrutura de repetição com o nome *Convert a byte and zero pad it*. O bloco no centro dela possui duas entradas uma são os 4 bytes de informação e a outra é o código %03u. Este bloco converte os 4 bytes recebidos em caracteres e codifica cada caractere com o valor da tabela ASCII. A saída deste bloco é conectada no bloco *VISA* que faz o envio do comando pela USB para a matriz.

Após receber o comando citado acima o *Matrix Controller* iria decodificar os doze bytes em quatro bytes (2, 96, 0, 0) recuperando assim a informação original. Com a informação decodificada o *Matrix Controller* envia três bytes com valores 96, 0, 0 respectivamente para o *Board Controller* com endereço 2 barramento I2C. O *Board Controller* da placa 2 então seta os relés 6 e 7.

3 *Melhorias feitas*

3.1 Visão Geral das Novas Características

Pelas razões citadas na seção 1.1 melhorias foram criadas na plataforma de experimentos remotos *VISIR Open Labs*. Para isto uma nova versão dos protocolos de comunicação usados foi criada e uma atualização no *firmware* dos micro controladores da matriz foi realizada. Para manter a mesma nomenclatura usada anteriormente no projeto a nova versão do protocolo continuará dividida em dois protocolos, o *Matrix Protocol* e o *Board Protocol*. Por isto tanto o *Matrix Protocol* quanto o *Board Protocol* foram atualizados. A nova versão dos dois protocolos tem como objetivo reportar erros para o *Equipment Server*. O *Matrix Protocol* tem como objetivo também controlar CI's conversores AD e DA. Esta atualização feita nos protocolos não tem como objetivo corrigir erros, apenas informá-los. As novas funcionalidades criadas com a atualização do *Board Protocol* e do *Matrix Protocol* são:

- Informar o status das placas conectadas a matriz.
- Informar a confirmação dos comandos executados.
- Tratamento de erros por *timeout*.
- Ler CI's AD .
- Escrever em CI's DA.
- Compatibilidade com a versão anterior.

O *Data Path* nessa nova versão é o mesmo mostrado na figura 2.9, mas agora o barramento digital tem mais pinos usados. Por definição do BTH o protocolo irá suportar até 30 placas na matriz, elas serão endereçados de 1 a 30. Para manter a compatibilidade os endereços da tabela 3.1 continuam reservados.

Endereço I2C	Tipo de placa	Adesivo
1	*Component Board* 1	COMP 1
2	*Component Board* 2	COMP 2
16	*Instrument Board* (conector para multímetro)	OSC 16
17	*Instrument Board* (conector para osciloscópio)	DMM 17
24	*Source Board*	SRC 24

Tabela 3.1: Esquema de endereços I2C

3.1.1 Metodologia de desenvolvimento

Por definição de projeto o novo protocolo também será dividido em dois. O *Matrix Protocol* que será usado na comunicação entre o *Equipment Server* e o *Matrix Controller* e o *Board Protocol* que será usado entre o *Matrix Controller* e os *Board Controller*.

O desenvolvimento do trabalho foi divido nas seguintes partes:

- Estudo da plataforma e protocolo já existente.
- Estudo do funcionamento do padrão USB.
- Implementação da comunicação em duas vias no *Matrix Controller*.
- Estudo do protocolo I2C.
- Estudo da linguagem assembly para a família PIC.
- Implementação do protocolo de comunicação entre as placas da matriz.
- Implementação do suporte a CI's AD e DA no barramento digital.
- Criação dos drivers em *LabView*.

Para desenvolver este trabalho as ferramentas citadas abaixo forma usadas:

- IDE *MPLAB*.
- Compilador C18.
- *Kit PICDEM 2 PLUS*.
- Gravador ICD2.
- IDE *LabView*.

MPLAB

É uma *Integrated Development Environment* (IDE) criada pela empresa *Microchip* para desenvolver o software para seus micro controladores. Uma imagem da IDE é mostrada na figura

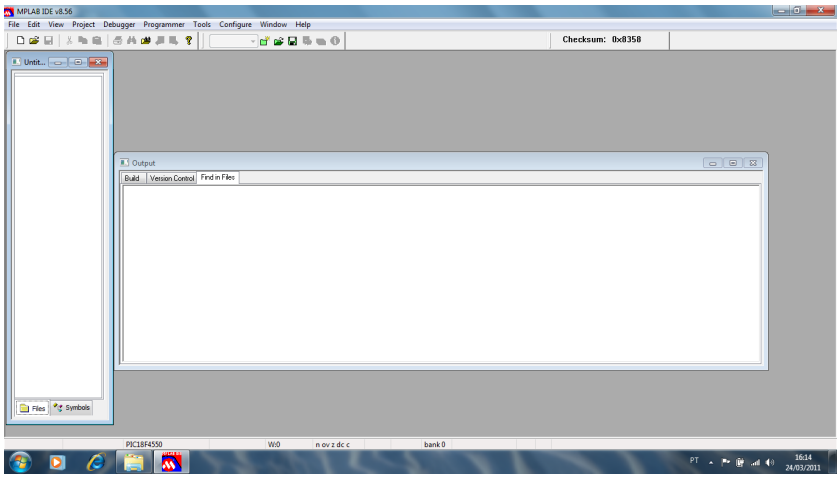

Figura 3.1: *IDE MPLAB*

Não há necessidade de licenças para o seu uso. Esta IDE oferece suporte para programação usando a linguagem *assembly* da família de micro controladores PIC e usando a linguagem C. Ela pode ser integrada com outros compiladores. No desenvolvimento deste trabalho foi usado a linguagem *assembly* para escrever o *software* dos micro controladores PIC16F767 e a linguagem C para escrever o código do micro controlador PIC18F4550. Ela oferece suporte também à simulação, mas nem todos os registradores podem ser simulados.

Compilador C18

Compilador criado pela empresa *Microchip* para a linguagem C, ele segue os padrões ANSI C. Ele foi usado, pois as bibliotecas da interface USB já usada foram escritas usando-o. Foi usado para compilar o código do micro controlador PIC18F4550.

Kit PICDEM 2 PLUS e gravador ICD2

O kit e o gravador foram usados para gravar o código hexadecimal gerado pelo compilador C18 nos micro controladores PIC18F4550. Este *kit* não possuía interface USB e não tinha suporte para adicionar outros dispositivos no barramento I2C. Isto implicava que não era possível *debuggar* o código.

LabView

LabView é uma linguagem de programação visual desenvolvida pela empresa *National Instruments* (INSTRUMENTS,). Programar em *LabView* consiste em selecionar blocos com funções já existentes, conectá-los e configurá-los. Ela permite também que o programador crie seus próprios blocos. O fato de ser uma linguagem de programação visual torna o desenvolvimento do software mais rápido. A sua IDE também é chamada de *LabView*. Esta linguagem foi usada para escrever os drivers da matriz para o *Equipment Server*.

3.2 *Matrix Protocol*

O novo *Matrix Protocol* define quatro comandos para a comunicação entre o *Equipment Server* e o *Matrix Controller*. Todos os novos comandos criados possuem as seguintes características:

- São enviados sempre do *Equipment Server* para o *Matrix Controller* através da USB.
- Geram uma resposta do *Matrix Controller*
- O primeiro byte de cada comando sempre é o cabeçalho.

Todos os comandos usados possuem 64 bytes, pois a interface USB usa a transferência do tipo *Bulk Transfer* para manter a compatibilidade com versões anteriores. Na descrição dos comandos serão considerados somente os bytes usados no comando. Os outros bytes possuem valor zero e devem ser desconsiderados. Os comandos e seus objetivos são descritos na tabela 3.2.

Cabeçalho	Comando	Descrição
0xFF	*MatrixStatusRequest*	Requisita o status das placas da matriz
0xFE	*CreateCircuit*	Envia dados para setar os relés e configurar o potenciômetro de uma placa da matriz
0xFD	*SendDigitalData*	Seta pinos do barramento digital conectados a um CI
0xFC	*ReadDigitalData*	Lê pinos do barramento digital conectados a um CI

Tabela 3.2: Novos comandos do *Matrix Protocol*

A descrição de cada comando é feita nas próximas subseções.

3.2.1 *MatrixStatusRequest*

Comando enviado quando o *Equipment Server* precisar saber a quantidade de placas, endereço das placas, tipo de placa e versão do *firmware*. Este comando é composto de um byte, somente o cabeçalho com valor 0xFF é enviado ao *Matrix Controller* conforme mostrado na tabela 3.3.

Byte 0
Header
0xFF

Tabela 3.3: Comando *MatrixStatusRequest*

O *Matrix Controller* trata o comando recebido, verificar todos os endereços do barramento I2C usando o *Board Protocol*. Esta operação será descrita na seção 3.3.2. Após verificar toda a matriz o *Matrix Controller* envia uma mensagem de reposta com 32 bytes contendo as informações desejadas. O formato da resposta é descrito na tabela 3.4.

Byte 0	Byte 1	Byte 2 - 31
Header	Byte de confirmação	Número das placas 1 à 30
0xFF	0x20	*Status Byte*

Tabela 3.4: Resposta do comando *MatrixStatusRequest*

O *Status Byte* citado acima contém o tipo de placa e a versão do *firmware*. Se o valor é zero, significa que não há nenhuma placa na matriz com aquele endereço. Quando o valor for diferente de zero os quatro bits mais significativos representam a versão de *firmware* e os quatro bits menos significativo representam o tipo de placa. A nova versão do *firmware* é 2 que é representado pela máscara 0010 nos quatro bits mais significativos. Os três valores para indicar o tipo de máscara são mostrados na tabela 3.5.

4 bits menos significativos	Valor	Tipo de placa
0001	0x01	*Component Board*
0010	0x02	*Instrument Board*
0011	0x03	*Source Board*

Tabela 3.5: Máscara de bits para os tipos de placa

3.2.2 *CreateCircuit*

Comando enviado para a matriz quando o estado de um relé ou do potenciômetro digital precisa ser alterado. O *Equipment Server* envia este comando pela USB para o *Matrix Controller*. O *Matrix Controller* trata o comando e encaminha os dados para a placa alvo pelo barramento I2C usando o novo *Board Protocol* que será descrito na seção 3.3.3. Este é um comando de cinco bytes. O primeiro byte é o cabeçalho com valor de 0xFE, o segundo o endereço da placa e os três últimos são as informações sobre os relés a serem alterados. Se a placa alvo for uma *Component Board* o último byte a resistência do potenciômetro digital. O formato do comando é mostrado na tabela 3.6.

	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4
Tipo de placa	Cabeçalho	Endereço da placa	Dados	Dados	Dados
Source Board	0xFE	Endereço da placa	Relés 1-7	Relés 8-14	Não usado
Instrument Board	0xFE	Endereço da placa	Relés 1-7	Relés 8-14	Relé 15 - 21
Component Board	0xFE	Endereço da placa	Relés 1-7	Relés 8-14	Conf potenciômetro

Tabela 3.6: Formato do comando *CreateCircuit*

Nos bytes que contem dados sobre os relés os sete bits menos significativos representam como o relé deve ser configurado e o bit mais significativo deve ser zero. Bit setado significa que o relé deve ser fechado, bit com valor zero significa que o relé não deve ser fechado. O bit 0 do byte 2 da mensagem corresponde ao relé 1, o bit 1 corresponde ao relé 2 e assim sucessivamente. Se o alvo for uma placa do tipo *Component Board* o último byte é a configuração da resistência do potenciômetro digital e por isto o bit mais significativo do último byte deve estar setado.

Após enviar o comando o *Equipment Server* espera por uma resposta do *Matrix Controller* informando a execução ou não do comando. Esta resposta possui dois bytes. O primeiro é o cabeçalho com valor 0xFE e o segundo é o byte de confirmação. O formato da resposta e os valores esperados do byte de confirmação são mostrados na tabela 3.7. Pode haver também erro por *timeout*, as respostas a erros de *timeout* é mostrado na subseção 3.2.5.

Cabeçalho	Byte de confirmação	Descrição
0xFE	0x20	Comando executado
0xFE	0x1	Comando não executado

Tabela 3.7: Resposta do comando *CreateCircuit*

3.2.3 *SendDigitalData*

Comando usado para enviar dados para 8 pinos do barramento digital. Os pinos usados para enviar esses dados são os pinos 1, 2, 3, 4, 7, 8, 9, 10 do barramento digital. Estes pinos devem estar conectados por fios no pino de entrada do CI conversor AD. Este comando não envolve o *Board Protocol* nem o barramento I2C, ele é executado somente pelo *Matrix Controller* situado na *Source Board*. O *Matrix Controller* escreve no barramento digital um byte. Este é um comando de dois bytes, onde o primeiro byte é o cabeçalho com o valor 0xFD e o segundo byte é a informação a ser escrita no barramento digital. Este comando é usado para escrever em CI's conectados ao barramento digital, como por exemplo, o conversor DA DAC0800LCN. O formato do comando é descrito na tabela 3.8.

Byte 0	Byte 1
Cabeçalho	Byte com informação
0xFD	0 - 255

Tabela 3.8: Formato do comando *SendDigitalData*

Após enviar o comando, o *Equipment Server* espera uma resposta com a confirmação do comando. A resposta é uma mensagem com dois bytes. A descrição dessa resposta é feita na tabela 3.9.

Cabeçalho	Byte de confirmação	Descrição
0xFD	0x20	Informação foi enviada para o barramento digital
0xFD	Outro valor	Informação não foi enviada para o barramento digital

Tabela 3.9: Resposta do comando *SendDigitalData*

3.2.4 *ReadDigitalData*

Este comando é usado para ler dados do barramento digital através dos pinos 1, 2, 3, 4, 5, 6, 7, 8, 9,10. Estes pinos devem estar conectados por fios na saída do CI conversor AD. Este comando é enviado pelo *Equipment Server* para o *Matrix Controller* através da USB. Este comando envolve somente o *Matrix Controller*, o barramento I2C e os *Board Controller* não são usados neste comando.

Quando o *Matrix Controller* recebe este comando ele configura também quatro pinos do barramento digital para controlar o CI conversor AD. Estes quatro pinos devem estar também conectados por fios ao CI conversor AD. Três destes pinos são configurados como pinos de

saída e com nível lógico alto. Um pino é configurado como entrada. Os pinos configurados como saída são os pinos 18, 19 e 20 do barramento digital. Eles são chamados de CS, RD e WR respectivamente.

A função de cada um desses pinos é:

- CS: Habilitar o *chip select* do CI.
- RD: Habilitar saída do CI.
- WR: Iniciar a conversão do sinal analógico para digital.

O pino configurado como entrada é o pino 21 que é chamado de INTR e tem a função de identificar quando o pino do CI ao qual ele está conectado mudar de nível lógico alto para nível lógico baixo. Isto indicará fim da conversão AD e que o *Matrix Controller* poderá ler o valor convertido pelo CI. Este funcionamento dos pinos de controle são compatíveis com o CI conversor AD ADC0804LCN. Caso o CI utilizado não seja o ADC0804LCN os pinos de controles não devem ser conectados ao CI conversor, exceto no caso que ele tenha o mesmo esquema de pinos de controle. Caso o CI utilizado que funcione no modo *stand alone* ou seja, não tenha pinos para controlar a conversão ele também poderá ser usado no circuito. Nesse caso os pinos CS, RD, WR e INTR não devem ser conectados. Este comando possui somente um byte, somente o cabeçalho com valor 0xFC é enviado ao *Matrix Controller* conforme mostrado na tabela 3.10.

Byte 0
Cabeçalho
0xFD

Tabela 3.10: Comando *ReadDigitalData*

Após a leitura o *Matrix Controller* responde com uma mensagem de três bytes, o primeiro é o cabeçalho, o segundo é o byte de confirmação e o terceiro é o valor lido. O formato da resposta é mostrado na tabela3.11.

Byte 0	Byte 1	Byte 2
Cabeçalho	Byte de confirmação	Valor lido
0xFD	0x20	0 - 255

Tabela 3.11: Resposta do comando *ReadDigitalData*

3.2.5 Tratamento de erros por *timeout*

Todos os comandos executados no barramento I2C usam loops para temporização, por alguma interferência eletromagnética ou por defeito no CI o *Matrix Controller* ou o *Board Controller* podem setar os bits errados ou não setar ou limpar alguns bits de controle. Isto iria gerar um *loop* infinito no *firmware* dos micro controladores envolvidos nesta transferência de dados. Para evitar este tipo de problema quando um comando é enviado no barramento I2C um temporizador é iniciado. O *timeout* do temporizador é de 350 ms para o comando *CreateCircuit* e de 900ms para o comando *MatrixStatusRequest*. Estes valores são cem vezes o tempo que cada comando levaria para ser executado em condições normais. Ao receber a resposta do comando enviado pelo barramento I2C o *Matrix Controller* para a contagem do temporizador. Se a resposta demorar mais tempo que o configurado no temporizador uma interrupção é gerada no software do *Matrix Controller*. Nessa interrupção uma mensagem de dois bytes é enviada ao *Equipment Server* informando o erro. Nesta mensagem o primeiro byte é o cabeçalho do comando que foi executado e gerou o time out e o segundo byte é o indicador de *timeout*. O formato da mensagem é mostrado na tabela 3.12.

Byte 0	Byte 1	
Cabeçalho do comando recebido	Indicador de *timeout*	Descrição
0xFF	0xFA	*Timeout* ocorrido enquanto era executado o comando *MatrixStatusRequest*
0xFE	0xFA	*Timeout* ocorrido enquanto era executado o comando *CreateCircuit*

Tabela 3.12: Mensagem de erro por *timeout*

3.2.6 Compatibilidade com a versão anterior

O novo *firmware* é compatível com a versão anterior do *software*. Detalhes da implementação desta compatibilidade não serão descritas nesse documento. A nova versão suporta todos os comandos da versão antiga descrita na seção 2.4. Como citado anteriormente os comandos antigos não geram resposta ou confirmação da execução do comando.

3.3 Novo *Board Protocol*

3.3.1 Visão geral

Este protocolo é usado pelo *Matrix Controller* e pelo *Board Controller* de cada placa na matriz. Todos os comandos deste protocolo possuem as seguintes características:

- São enviados pelo *Matrix Controller* para o *Board Controller* da placa alvo.
- O primeiro byte do comando é o cabeçalho.
- O *Board Controller* sempre gera alguma resposta ao *Matrix Controller*.

Dois comandos foram definidos, o Board Circuit Command e o Board Status Command. A descrição deles é feita abaixo:

3.3.2 Board Status

Este comando é usado quando o *Matrix Controller* recebe o comando *MatrixStatusRequest* do *Equipment Server*. O objetivo deste comando é informar ao *Matrix Controller* a versão do *software* e o tipo de placa. Este comando possui três bytes. Ele é enviado para as placas com endereço de 1 a 30. Por definição o número máximo de placas em cada matriz é 30. Para enviar o endereço da placa no barramento I2C, o endereço deve ser multiplicado por dois para garantir que seja um valor par. Isto ocorre pois segundo a especificação do protocolo I2C os sete bits mais significativos devem conter o endereço e o bit menos significativo informa se o mestre do barramento esta escrevendo ou lendo o escravo. Neste caso o mestre esta escrevendo no escravo então o bit menos significativo tem valor zero. Quando o mestre ler algum valor do escravo o endereço será multiplicado por 2 também, mas o bit menos significativo terá valor um.

O *Matrix Controller* envia o primeiro bit com o endereço da placa multiplicado por dois nos sete bits mais significativos e o bit menos significativo zerado. O segundo byte é o cabeçalho com valor 0xFF. Após enviar o cabeçalho o *Matrix Controller* gera um

restart no barramento I2C e envia o terceiro byte com o endereço multiplicado por 2 nos sete bits mais significativos e o bit menos significativo setado. Isto informa ao *Board Controller* que o *Matrix Controller* esta esperando receber um byte. O formato do comando é mostrado na tabela 3.13.

A resposta dada pelo *Board Controller* da placa alvo é um byte contendo a versão do *firmware* e o tipo de placa. Os quatro bits mais significativos representam a versão do *firmware*

Byte 0	Byte 1	*Restart* no barramento I2C	Byte 2
Endereço da placa multiplicado por 2	Cabeçalho 0xFF		Endereço da placa multiplicado por 2 e adicionado 1

Tabela 3.13: Formato do comando Board Status

no *Board Controller*. A versão atual é dois, então os quatro bits mais significativos devem ter o valor 0010. Os quatro bits menos significativos informam o tipo de placa. Os valores possíveis nos quatro bits menos significativos são mostrados na tabela 3.14.

Quatro bits menos significativos	Valor	Tipo de placa
0001	0x01	*Component Board*
0010	0x02	*Instrument Board*
0011	0x03	*Source Board*

Tabela 3.14: Máscara de bits para os tipos de placa

3.3.3 Board Circuit

Comando usado quando o *Matrix Controller* receber um comando do tipo *CreateCircuit*. Este comando é usado para alterar o estado de relés. Este comando possui seis bytes. O primeiro é o endereço da placa multiplicado por dois, o segundo byte é o cabeçalho com valor 0xFE, do terceiro ao quinto byte são bytes com dados dos relés. O quinto byte pode ser a configuração da resistência do potenciômetro se for uma *Component Board* ou pode não ser usado se for uma *Source Board*. Após o quinto byte o *Matrix Controller* gera um *restart* no barramento I2C(MICROCHIP, a) e o sexto e último byte é endereço da placa multiplicado por dois e somado mais um. Isto ocorre para o *Matrix Controller* informar ao *Board Controller* está esperando receber um byte. O formato deste comando é mostrado na tabela 3.15.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	I2C *restart*	Byte 5
Endereço	Cabeçalho	Dados	Dados	Dados		Endereço
Número da placar x 2	0xFE	Relé 1 - 7	Relé 8 - 14	Relé 15 - 21 ou potenciômetro ou não usado		Número da placa x 2 + 1

Tabela 3.15: Formato do comando Board Circuit

O conteúdo dos bytes com informações dos relés é o mesmo citado na seção 3.2.2.

Após energizar os relés o *Board Controller* envia um byte com a confirmação da execução do comando para o *Matrix Controller*. Os valores descritos são descritos na tabela 3.16.

Byte de confirmação	Descrição
0x20	Comando executado
0x1	Comando não executado

Tabela 3.16: Resposta do comando Board Circuit

3.4 Drivers em LabView

Para que o *Equipment Server* possa comunicar com a matriz foi necessário escrever drivers para cada novo comando executado. Os drivers criados simplesmente organizam os bytes no formato do comando, escrevem e depois lêem a interface USB. Em todos os drivers é feito a verificação de erros da interface USB. Um dos exemplos é o driver para o comando *CreateCircuit*. Este driver tem a função de receber os bits com informação sobre os relés em um *array* de bytes e insere o cabeçalho na primeira posição do *array* e o endereço da placa na segunda posição do *array*. O formato dos bytes a serem enviados é mostrado no visor *Command Sent* e também é convertido em *string*. Após ser convertido em *string* ele é enviado para o bloco mostrado *Write USB* conforme mostra a figura 3.2.

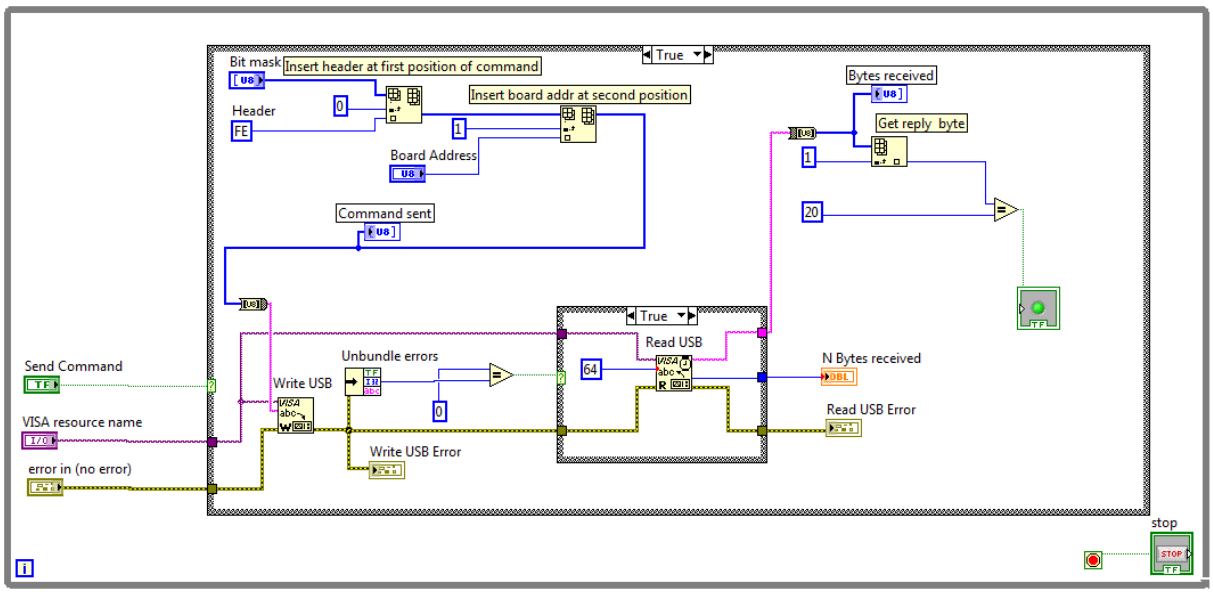

Figura 3.2: *Driver* do comando *CreateCircuit*

Além do comando a ser enviado o bloco *Write USB* também recebe *VISA resource name* que neste caso é a matriz conectada pela USB. Após a escrita na USB é verificado a quantidade de erros da interface USB no *Unbundle Errors* o número do erro é enviado para o *Write USB Error*. Se o número de erros for zero é feito a leitura da USB no bloco *Read USB* que além do *VISA resource name* recebe também a quantidade de bytes a serem lidos pela USB. Neste caso são lidos 64 bytes para que a configuração da USB permanecesse compatível com a versão anterior. O bloco *Read USB* tem três saídas. Elas são, a informação lida na USB em formato *string* que depois é convertida para byte *array*, o número de bytes lidos e informação dos erros da USB. No byte *array* gerado é verificado se o segundo byte tem valor 0x20 para informar se o comando foi executado ou não.

Todos os comandos do *Matrix Protocol* seguem a mesma organização dos blocos, mudando apenas detalhes no byte *array* que será enviado pela USB. Figuras com os *drivers* dos outros comandos são mostradas abaixo.

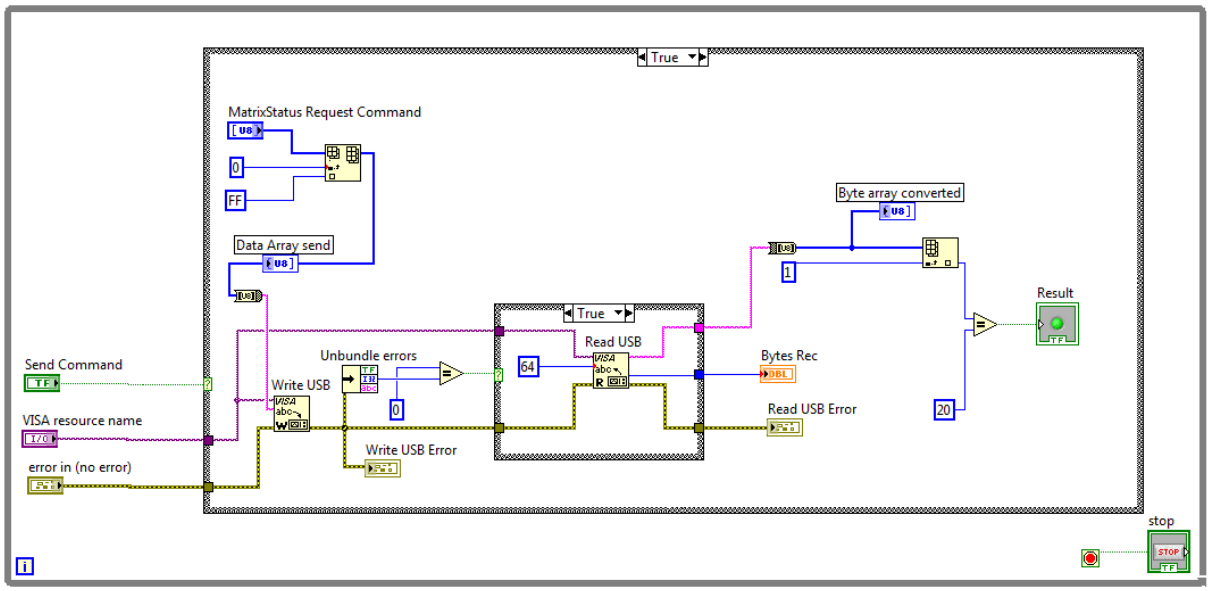

Figura 3.3: *Driver* do comando *MatrixStatusRequest*

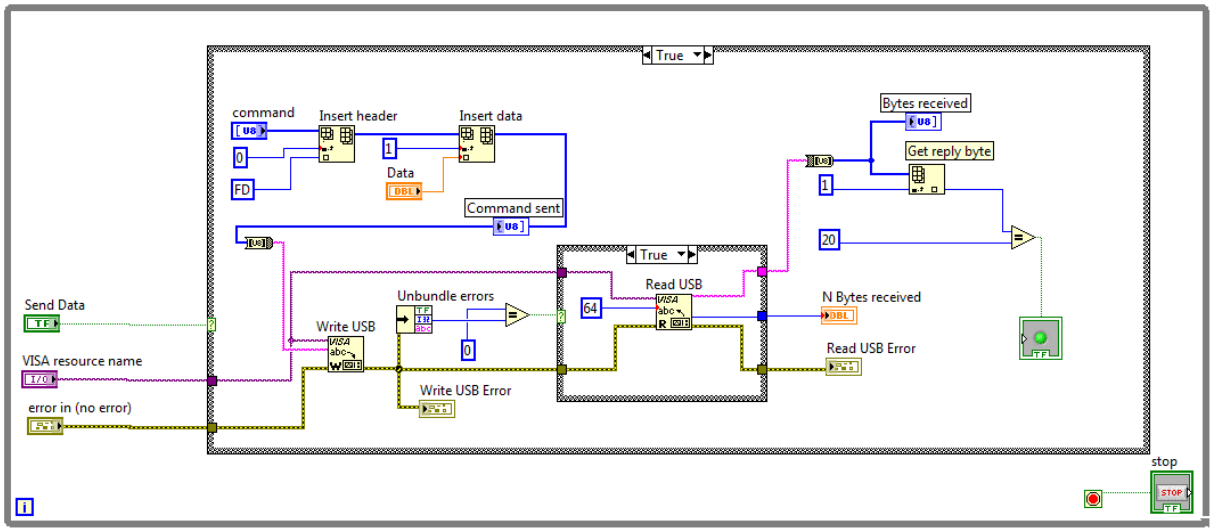

Figura 3.4: *Driver* do comando *SendDigitalData*

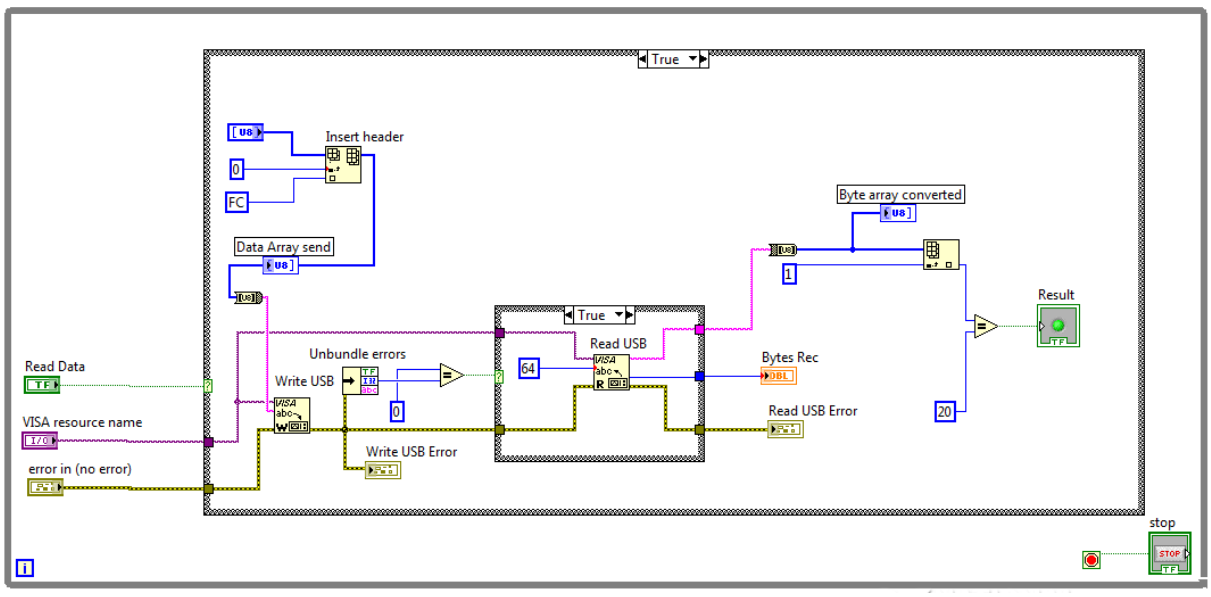

Figura 3.5: *Driver* do comando *ReadDigitalData*

4 Exemplos e Comparações

Neste capítulo serão mostrados dois exemplos do uso da nova versão do protocolo e comparações com a versão anterior.

4.1 Exemplos

Esta seção mostra dois exemplos do uso da nova versão do *software*. O primeiro é requisitar o status da matriz usando o comando *MatrixStatusRequest* e o segundo é a leitura de um CI conversor analógico para digital ADC0804LCN.

4.1.1 Requisitar o status da matriz

Para requisitar o status da matriz o *Equipment Server* deve enviar um comando *MatrixStatusRequest*. Após receber este comando o *Matrix Controller* ira enviar um Board Status para o primeiro endereço de placa, neste caso o endereço 1. Se houver placa neste endereço ela irá responder com um byte informando o tipo de placa e a versão do *firmware*. Após repetir o procedimento para os endereços 1 ao 30 o *Matrix Controller* irá enviar uma mensagem com 32 bytes para o *Equipment Server* conforme mostrado na tabela 3.4. A figura 4.1 mostra o caminho da mensagem enviado.

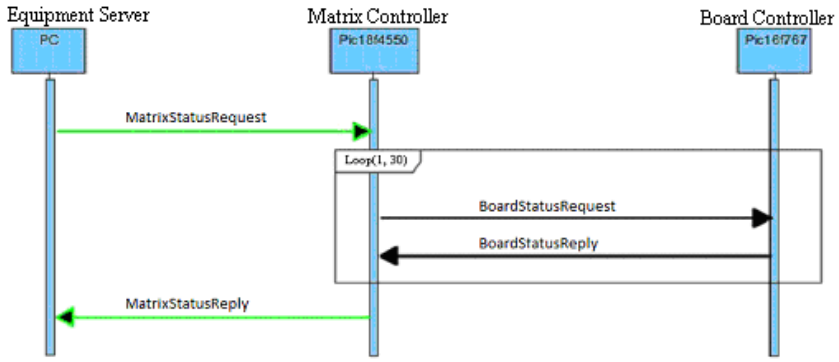

Figura 4.1: Caminho das mensagens em um *MatrixStatusRequest*

Um driver em *LabView* pode ser usado no *Equipment Server* para enviar este comando, um exemplo de driver em *Labview* para essa função é mostrado na figura4.2.

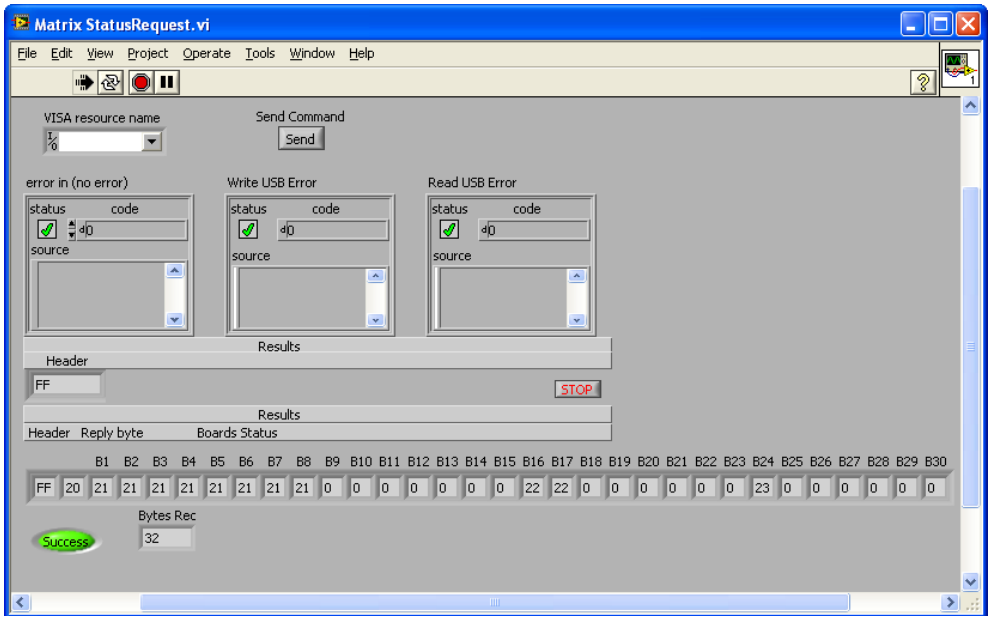

Figura 4.2: Programa em *Labview* para executar um *MatrixStatusRequest*

Na figura 4.2 é mostrado o cabeçalho do comando enviado e a resposta com 32 bytes. Na resposta, da placa número 1 (B1) até a placa número 8 (B8) o byte resposta tem valor 21. Se separarmos este byte em dois grupos de quatro bits veremos que os quatro bits mais significativos possuem valor 02 que significa que a versão de *software* usada é 2. Veremos também que os quatro bits menos significativos possuem valor 1, indicando que é uma *Component Board*. Na placa 16 e 17 (B16 e B17) o valor do byte de resposta é 22. Isto significa que essas duas placas usam o software na versão 2 e são Equipment Board. Na placa 24 (B24) o valor de resposta é 23, isto significa que a versão do *firmware* é 2 e é uma *Source Board*.

4.1.2 Leitura do conversor AD ADC0804LCN

Com o suporte à leitura e escrita no barramento digital é possível controlar o CI ADC0804LCN. Este conversor AD possui quatro pinos para controlá-lo. É possível usar o comando *ReadDigitalData* para ler o valor convertido em digital. Uma opção para as conexões com fios para este CI é mostrado na figura4.3.

Na figura 4.3 os fios vermelhos conectados ao barramento digital são usados para controlar o CI. Os fios azuis conectados ao barramento digital são usados para fazer a leitura da saída do CI. Os fios amarelos conectados aos nós C e D são as sinais analógicos provenientes do gerador de funções. Para este exemplo dois comandos devem se enviados. O primeiro seria um

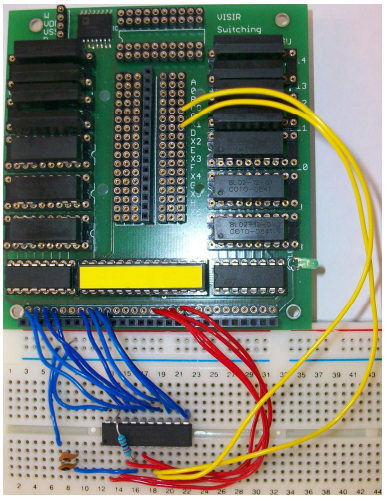

Figura 4.3: Conexão do CI ADC0804LCN com o barramento digital

CreateCircuit para a Source Board conectar os nós C e D ao gerador de funções. O segundo seria o *ReadDigitalData* para controlar os quatro pinos do CI ADC 0804LCN e ler a sua saída. Após a leitura o *Matrix Controller* envia uma resposta ao *Equipment Server* com o valor lido em seu terceiro byte conforme mostra a tabela 3.11. Para executar a leitura do barramento digital um programa em *Labview* conforme mostrado na figura 4.4 poderia ser usado.

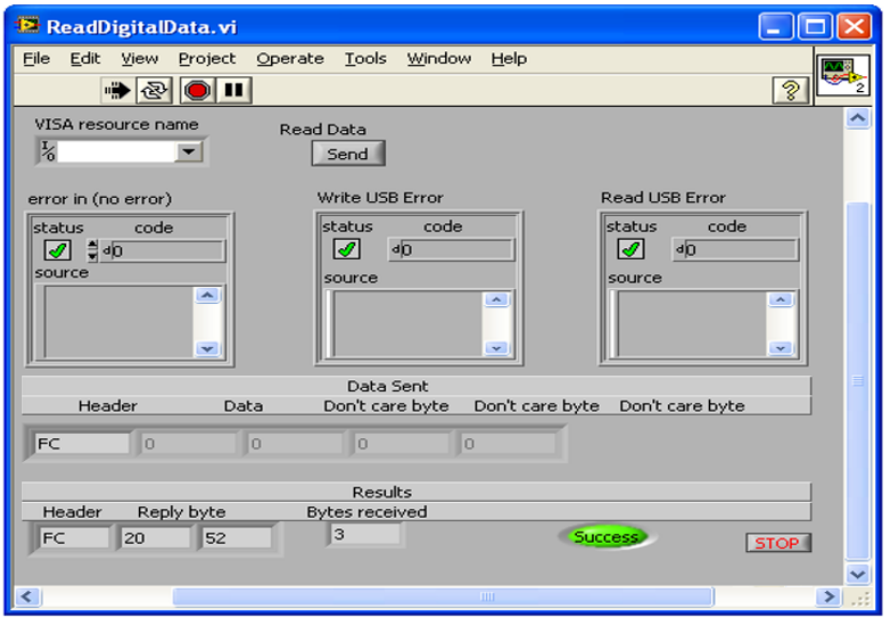

Figura 4.4: Programa para executar o *ReadDigitalData*

4.2 Validação

Para validar o funcionamento da nova versão do protocolo diferentes testes foram executados para validar o software. Os testes feitos foram:

- Compatibilidade com versão anterior;
- Exaustividade de comandos;
- Estado dos relés;
- Escrita em conversores DA;
- Leitura de conversores AD;

A descrição simples de cada um deles é feita abaixo:

4.2.1 Compatibilidade com versão anterior

Um dos requisitos da nova versão do firmware dos micro controladores era que a matriz continuasse aceitando comandos da versão anterior do protocolo. Para validar estes requisitos um teste dividido em duas etapas foi feito. A primeira etapa foi validar o novo *firmware* do *Matrix Controller*. Uma *Source Board* com o novo firmware foi usada para controlar uma matriz onde todos os *Board Controllers* usavam a versão anterior de *firmware*. Esta matriz foi responsável por executar experimentos de uma turma de estudantes de engenharia elétrica do BTH por uma semana. Durante esse período nenhum problema foi encontrado. A segunda parte foi atualizar o *firmware* para a nova versão em todos os *Board Controllers* desta matriz. Após a atualização a matriz foi usada por para executar experimentos de uma segunda turma de estudantes do BTH durante uma semana. Um problema foi detectado no tratamento de interrupções dos *Board Controller*. Este problema foi resolvido reescrevendo tratador de interrupções do *Board Controller*. A matriz com a nova versão é a usada atualmente nos experimentos realizados no BTH.

4.2.2 Exaustividade de comandos

Como há uso de *timers* e interrupções na comunicação entre as placas, e a plataforma é utilizada por vários alunos simultaneamente, um teste para avaliar o novo software com uma alta carga de trabalho era necessário. Este teste consistia enviar o mesmo comando repetidamente.

Foi criado um programa em C++ para fazer o envio do comando *CreateCircuit* e do *MatrixStatusRequest* através de linha de comando. O programa envia o comando selecionado e mostra a resposta na tela. Se o valor da resposta fosse sempre igual mostrava que o *firmware* do *Matrix Controller* e do *Board Controller* não apresentavam problemas em executar uma alta carga de trabalho. Para gerar essa alta carga de trabalho o envio do comando e leitura da resposta foi executado 1000 vezes em seguida. A figura 4.5 mostra uma parte do código do programa que enviava estes comandos.

```
/*
 d is the usb target device
 n is the number of repetitions
*/
void send_new_protocol_command(usb_dev_handle * d, int n){
   int addr; //Board address
   int j ;

   printf("\nType the board address:\n");
   cin  >> addr;

   for( j = 0; j < n ; j++){
      //Clean array
      for( int i = 0 ; i < reqLen ; i++ ){
         question[i] = 0;
         rec[i] = 0;
      }
      question[64] = '\0';
      rec[64] = '\0';

      question[0] = 0xFE; //Header
      question[1] = addr;
      question[2] = 0x7F; //Set all relays
      question[3] = 0x7F;
      question[4] = 0x7F;

      send_usb(d , reqLen, question); // send command
      recv_usb( d , reqLen , rec ); // read command reply

      printf("\nBYTE:%d REC:%X\n",j,(byte)rec[0]);
   }
}
```

Figura 4.5: Função usada para enviar o comando *CreateCircuit* repetidamente para a matriz

Após o envio e leitura de 1000 comandos em seguida foi verificado que a nova versão do firmware era estável e suportava uma carga alta de trabalho. Isto é algo extremamente importante pois uma única matriz iria receber comando para executar experimentos de vários estudantes.

4.2.3 Estado dos relés

O comando *CreateCircuit* tem como finalidade montar o circuito criado pelo estudante alterando o estado dos relés. O teste para validar o estado dos relés consta em enviar um comando para uma placa e verificar que os relés corretos foram fechados ou abertos. Para testar o estado de cada relé foi usado o programa *matrixapp*. Ele também foi alterado para receber a resposta dos comandos efetuados. A tela do *matrixapp* é mostrada na figura 4.6.

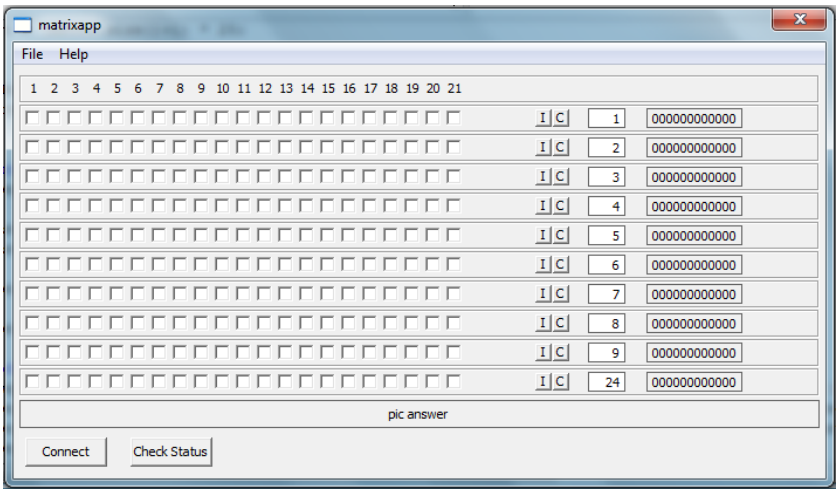

Figura 4.6: Tela do software de teste *matrixapp*

Na tela do software mostrado na figura 4.6 as colunas de 1 a 21 são o endereço dos relés, as linhas com valores de 1 a 9 e 24 são o endereço das placas e os painéis ao seu lado mostram o formato da organização da informação original dos bytes antes de ser codificada em ASCII. Para enviar um comando era necessário selecionar a linha que possua o endereço da placa alvo e depois setar os quadrados informando o estado dos relés que você irá alterar. Se fosse marcado significava que o rele deveria ser fechado, se não fosse marcado significava que o relé deveria ser aberto. Com isto era possível setar cada relé na placa. Após selecionar o relé a ser fechado era necessário verificar a continuidade entre os terminais do relé usando um multímetro. Esta verificação com o multímetro era feita em cada um dos relés de cada tipo de placa. Este teste foi realizado para cada um dos relés de cada tipo de placa.

4.2.4 Escrita em conversor DA

Este teste foi feito para validar o comando *SendDigitalData*. Para a sua execução foi usado o programa *command*. O procedimento consistia em enviar um byte para o CI conversor DA e medir a sua saída analógica. O CI usado foi o DAC0800LCN. Através das conexões de seus pinos ele foi configurado para que a tensão de saída variasse entre -9.920 e 10 volts . O passo

de conversão era de 77.8 milivolts. A saída deveria ter o valor de:

$$byteenviado \cdot 77mV$$

. Os bytes enviados tinham valores de 0x00 até 0xFF. Para todos os 256 bytes enviados os valores de saída apresentados do CI foram o esperado mostrando o correto funcionamento do comando *SendDigitalData*.

4.2.5 Leitura em conversor AD

Este teste foi feito para testar o comando *ReadDigitalData*. Para a sua execução foi usado o programa *command*.O procedimento consistia em variar a tensão analógica de entrada do CI conversor AD e ler um byte em sua saída. O CI conversor usado foi o ADC0804LCN. Através das conexões de seus pinos ele foi configurado para usar um passo de conversão de 20 mV. Durante os testes a tensão analógica de entrada variou de 0 a 5 volts em passos de 20 mV. Cada vez que a tensão era alterada uma leitura dos pinos de saída do CI era feita. Para todos os valores o resultado foi o esperado.

4.3 Comparações

Após o desenvolvimento do software dos micro controladores e validação das novas funcionalidades, uma comparação entre a nova versão e a sua antecessora é válida. Serão feitas duas comparações entre as versões, uma é referente ao uso de memória dos micro controladores e outra é entre as funcionalidades.

4.3.1 Uso de memória

A comparação do uso de memória será feito com base nos no consumo da memória de programa e de dados mostrados pela IDE MPLAB. Na figura 4.7 é mostrada uma comparação do uso de memória de dados e de programa entre a antiga e a nova versão do *software* do *Board Controller*. A antiga versão é a tela da esquerda e a da direita é a versão nova. É notado na figura que foram usado somente 72 bytes de memória de programa a mais na nova versão. A memória de dados continua a mesma. Somente 4 por cento da memória de programa esta sendo usada. Isto permite que ainda novas versões de *software* possam inserir mais linhas de código sem ter o tamanho do código como uma dos grandes problemas. Para manter o baixo uso da memória de programa e de dados muitas partes do código que não eram úteis mais foram apagadas e outras

foram otimizadas.

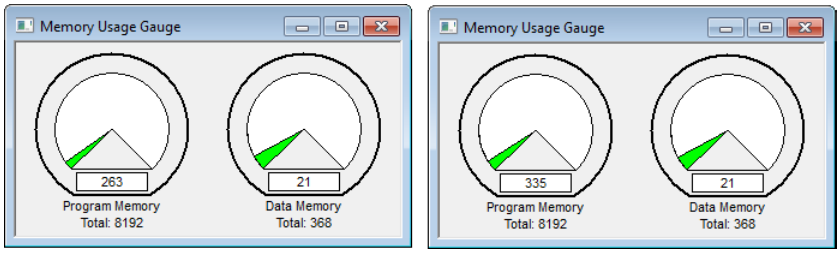

Figura 4.7: Comparação do uso de memória do PIC16F767

Na figura 4.8 é mostrado uma comparação do uso de memória de programa e de dados entre a antiga e nova versão. A versão antiga é mostrada na esquerda e o da direita mostra a nova versão. Como o PIC18F4550 tem mais recursos que o PIC16F767 o seu *software* foi escrito em linguagem C. Isto leva a um consumo maior de memória, mas torna o desenvolvimento do *software* mais rápido. A nova versão do software consome 680 bytes a mais de memória de programa e 19 byte a mais de memória de dados. Mesmo com a pilha do protocolo USB integrada e funções para o uso da I2C a nova versão usa somente 13.5 por cento da memória de programa e 25.92 por cento da memória de dados do micro controlador. Isto permite que no futuro muitas linhas de código possam ser adicionadas sem ter grandes problemas com o tamanho do código.

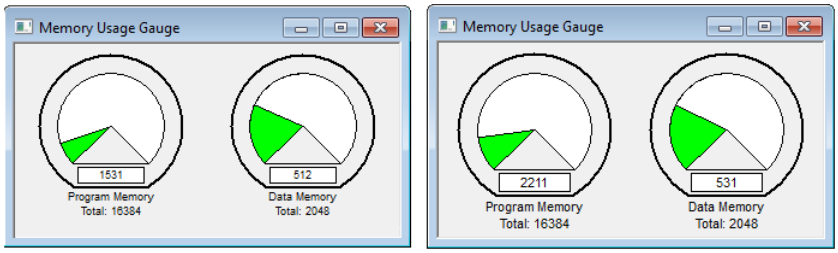

Figura 4.8: Comparação do uso de memória do PIC18F4550

4.3.2 Funcionalidades

Ao comparar as funcionalidades entre as duas versões verificamos que há uma grande diferença na quantidade. A lista abaixo mostra as funcionalidades da versão antiga.

- Criar circuitos

A lista abaixo mostra as funcionalidades da versão nova.

- Criar circuitos
- Informar o status das placas conectadas a matriz.
- Informar a confirmação dos comandos executados.
- Tratamento de erros por time-out.
- Ler CI's AD.
- Escrever em CI's AD.

5 *Conclusões*

Neste trabalho foi apresentado as melhorias feitas em parte da plataforma de experimentos remotos *VISIR Open Labs*. Foi descrito e implementado um novo protocolo a comunicação entre as placas da matriz, bem como o seu controle feito pelo *Equipment Server*. Este novo protocolo criou novas funcionalidades para a plataforma. As novas funcionalidade foram descritas no capítulo 3. Após o desenvolvimento e testes desta nova versão foi notado que as novas funcionalidades da plataforma serão muito úteis. Com a nova versão é possível ter confirmação de comando executados, verificar o status da matriz, ler e escrever no barramento digital. A plataforma está mais confiável e mais simples para gerenciar. Isto economiza tempo do professor ou funcionário que prepara os experimentos. Isto pode facilitar na expansão do número de universidades que utilizam esta plataforma e torna o estudo da engenharia mais fácil e com menor custo.

Durante o estudo da plataforma e desenvolvimento do novo software algumas limitações e dificuldades foram encontradas. A solução destas limitações e dificuldades são indicadas como trabalhos futuros, elas são:

- **Expansão das melhorias criadas da matriz:** O trabalho descrito neste documento implementou as novas funcionalidade citadas no capítulo 3 apenas na comunicação entre as placas da matriz e entre a matriz e o *Equipment Server*. Estas funcionalidades ainda não estão disponíveis para os aluno. Uma idéia de trabalho futuro seria alterar o código do *Measurement Server* e do *Experiment Client* para suportarem o uso de CI's conversores AD e DA. Isto tornaria as facilidades criadas disponíveis para os estudantes. Para complementar esta idéia de trabalho futuro, o código do *Equipment Server* poderia ser alterado para que quando a matriz informasse algum erro ele enviasse um e-mail ou SMS para o gerente da plataforma.

- **Sistemas de testes:** Durante o desenvolvimento do *software* para testar as funcionalidades do *software* era necessário setar cada relé um por vez e fazer o teste de continuidade usando o multímetro. Isto tomou muito tempo, por isso notou-se que seria necessário

montar uma ferramenta para validar as funcionalidades do *Matrix Controller* e do *Board Controller* e testar quais relés são energizados. Isto tornaria a validação do sistema muito mais rápida. Poderia ser criado também uma plataforma para testes de estabilidade, comportamento perante diferentes cargas de trabalho, performance dos equipamentos e comportamento perante interferência eletromagnética e ruídos.

Lista de Abreviaturas

ASB Departamento de Processamento de Sinais

AD Analógico Digital

ASCII *American Standard Code for Information Interchange*

BTH *Blekinge Institute of Technology*

CI Circuito Integrado

DA Digital Analógico

HTML *Hypertext Markup Language*

IDE *Integrated Development Environment*

I2C *Inter-Integrated Circuit*

PCI *Peripheral Component Interconnect*

SCL *Serial Clock*

SDA *Serial Data Line*

USB *Universal Serial Bus*

XML *Extensible Markup Language*

Referências Bibliográficas

GUSTAVSSON, I. *How to open a local electronics laboratory for remote access*. [S.l.]: International Association of Online Engineering, 2008.

GUSTAVSSON, I.; ZACKRISSON, J. The VISIR project–an open source software initiative for distributed online laboratories. *Proceedings of the REV* . . . , 2007. Disponível em: <http://www.bth.se/fou/forskinfo.nsf/all/2504f1d4ab273565c1257321004839ad/$file/54_paper.pdf>.

INSTRUMENTS, N. *Labview user manual*. Disponível em: <www.ni.com/pdf/manuals/320999e.pdf>.

MICROCHIP. *Application note AN735*. Disponível em: <www.ni.com/pdf/manuals/320999e.pdf>.

MICROCHIP. *PIC18F4550 Datasheet*. Disponível em: <ww1.microchip.com/downloads/en/DeviceDoc/39632e.pdf>.

NXP. *I2C Bus especification*. Disponível em: <www.nxp.com/acrobat_download2/literature/9398/39340011.pdf>.

Tor A. Fjeldly, M. S. S. *Lab on the Web: Running Real Electronics Experiments via the Internet*. [S.l.]: John Wiley, 2005.

USB.ORG. *USB Bus especification*. Disponível em: <http://www.usb.org/developers/docs>.