dx.doi.org/10.7437/NT2236-7640.2011.02.001
Notas Técnicas, v. 1, n. 2, 2011

Projeto e Teste de um Circuito de Controle Baseado no Barramento CAN – *Controller Area Network*

R. G. Gama,* A. F. Barbosa, e H. P. Lima Jr
Centro Brasileiro de Pesquisas Físicas - CBPF
Rua Dr. Xavier Sigaud,
150 – Urca – Rio de Janeiro – RJ – Brasil

A presente nota técnica apresenta o desenvolvimento de um protótipo de circuito de controle – *slow-control* – para um módulo de aquisição e digitalização de sinais que será utilizado no Projeto Neutrinos Angra, projeto coordenado pelo CBPF. O *slow-control* reúne funcionalidades que permitem a configuração e a monitoração de diversos parâmetros do módulo de forma online, sem interrupção da leitura de dados relativos. O projeto apresentado prevê que a comunicação com o circuito de controle será feita através de um barramento serial baseado no protocolo CAN – Controller Area Network. Tal protocolo garante a integridade dos dados através de técnicas robustas de correção de erro. Reunindo as características acima, o circuito de controle *slow-control* permite que um operador configure e depure o módulo eletrônico de aquisição de dados em campo, a partir de um computador pessoal instalado remotamente. Este documento descreve o projeto do circuito de controle, seu funcionamento e apresenta duas aplicações do protótipo: testes para comprovar a eficiência da comunicação e controle de um conversor digital-analógico através de um computador pessoal remoto.

1. INTRODUÇÃO À AQUISIÇÃO E DIGITALIZAÇÃO DE SINAIS EM EXPERIMENTOS DE FÍSICA

Experimentos de física de altas energias utilizam diversos dispositivos eletrônicos, que geralmente têm a finalidade de tratar os sinais elétricos provenientes de detectores que por sua vez traduzem um determinado evento físico em uma quantidade de carga elétrica, como ilustra a Figura 1. Esta quantidade de carga elétrica é amplificada e condicionada pela etapa Eletrônica de *front-end*, onde é transformada em um sinal elétrico com características como amplitude, forma e duração, adequadas ao funcionamento da próxima etapa – o Sistema de aquisição de dados. Este sistema pode integrar diversos dispositivos que trabalham em conjunto para adquirir e digitalizar os sinais analógicos de entrada. Os sinais digitalizados são essencialmente reproduções das formas de onda dos sinais gerados no detector de eventos, mas também podem ser o resultado de um processamento em tempo real do sinal digitalizado, de acordo com as necessidades específicas de cada experimento. Normalmente, a última etapa é realizada por um computador pessoal ou uma rede de computadores e servidores de informação, de forma que os dados resultantes da etapa anterior possam ser armazenados e processados posteriormente [1].

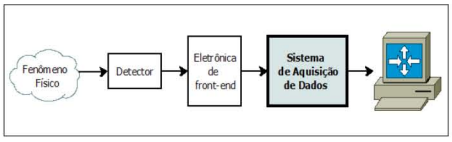


Figura 1: Aparato eletrônico típico para detecção de fenômenos físicos.

Pode-se destacar entre os dos dispositivos eletrônicos que compõem o Sistema de aquisição de dados o módulo eletrônico responsável pela aquisição e digitalização de sinais elétricos. O canal de comunicação estabelecido entre o módulo de digitalização de sinais e o restante do sistema de aquisição de dados e costuma ser escolhido de acordo com a taxa de dados que deve trafegar pelo mesmo, que é principalmente uma função da taxa de eventos físicos no experimento. Certificar-se da integridade da comunicação com um módulo eletrônico como o referido é de grande importância em um experimento científico, não somente pela importância da informação gerada pelos eventos físicos, mas também pela necessidade de calibração e depuração do próprio módulo.

Em certos experimentos, os eventos físicos podem ser tão abundantes que a taxa de dados gerada poderia sobrecarregar os meios de comunicação mais rápidos já desenvolvidos. Portanto, qualquer interrupção no fluxo de dados que não faça parte do próprio protocolo de comunicação deve ser eliminada. Tal objetivo pode ser atingido fazendo-se com que o canal de comunicação dos dados digitalizados seja dedicado apenas a esta função. Portanto, o circuito de controle para calibração e depuração do módulo de digitalização deve possuir um canal de comunicação independente do anterior. O próximo tópico desta nota técnica apresenta um circuito de controle para um módulo eletrônico de aquisição e digitalização de sinais. Este circuito de controle atende o citado requisito para o meio de comunicação.

2. UM CIRCUITO DE CONTROLE PARA UM MÓDULO DE AQUISIÇÃO E DIGITALIZAÇÃO DE SINAIS

O circuito de controle apresentado nesta nota técnica foi projetado para implementar meios de configuração, calibração e depuração para o módulo de aquisição e digitalização de sinais – NDAQ – que será utilizado no Projeto Neutrinos Angra – experimento para detecção de antineutrinos oriundos do reator de Angra II [2]. O circuito de controle possui um meio de comunicação dedicado, o barramento CAN [3], visando robustez e eficiência nas operações disponíveis. Através desta característica, descarta-se a necessidade de interromper o fluxo de dados digitais através do barramento principal de comunicação com o módulo. Este módulo eletrônico poderá operar em dois modos: como um módulo VME – padrão de eletrônica modular [4], onde o

*Electronic address: rgama@cbpf.br

meio de comunicação principal é o barramento VME e no modo *standalone*, onde o meio de comunicação principal é USB – barramento serial universal [5].

O módulo NDAQ pode ser visto como um dispositivo de processamento de dados que reúne as seguintes funcionalidades: 8 canais de conversão analógico-digital, 8 canais de conversão tempo-digital e 8 canais de circuitos discriminadores de sinal (que disparam os canais de conversão tempo-digital). Foi projetado de acordo com o padrão VME64, é baseado em dispositivos lógicos programáveis do tipo FPGA e utiliza componentes eletrônicos disponíveis comercialmente, como: conversores analógico-digital de alta velocidade, conversores tempo-digital, microcontroladores, memórias do tipo FIFO (*first-in, first-out*) e memórias RAM estáticas. O diagrama em blocos ilustrado na Figura 2 apresenta os principais componentes deste módulo. Os componentes identificados por CLOCK – distribuidor de *clock* e ADC – conversor analógico-digital, assim como os FPGAs – dispositivo lógico programável do tipo *Filed Programmable Gate Array*, responsáveis pelo núcleo do sistema, identificado por CORE, e pela interface com os barramentos de comunicação, identificado por VME, possuem porta de comunicação SPI – *Serial Peripheral Interface* [6]. Esta porta permite que registradores internos destes componentes sejam configurados e também, que informações sobre o estado de cada um destes componentes sejam verificadas.

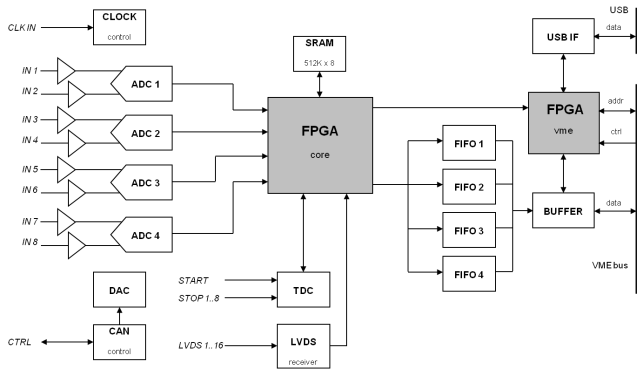


Figura 2: Diagrama em Blocos do módulo eletrônico digitalizador de sinais – NDAQ.

Além dos componentes já referidos, o NDAQ possui dois dispositivos que devem ser configurados através da comunicação SPI, estes são conversores digital-analógico, identificados por DAC. Um deles é responsável por gerar a tensão de *threshold* para os circuitos discriminadores de sinais e o segundo por gerar a tensão de *offset* para os circuitos pré-amplificadores de sinais, bloco que condiciona os sinais analógicos de entrada para serem entregues aos ADCs.

O circuito de controle é baseado em um dispositivo microcontrolador que integra periféricos de comunicação SPI e CAN. O programa executado no microcontrolador promove a interface entre as portas SPI dos dispositivos do NDAQ e o barramento CAN, através da interpretação de comandos encapsulados nas mensagens que trafegam pelo barramento CAN. As operações disponíveis dependem de uma aplicação de controle executável em um computador pessoal, que deve estar conectado ao referido barramento. A aplicação (*software*) e o circuito (módulo NDAQ) formam um sistema de controle que tem os seguintes requisitos: i) a aplicação de controle deve controlar diversos módulos NDAQ, ii) garantia de entrega da informação, iii) operação em longa distância sem repetidor e iv) baixo custo. As informações transferidas (leitura ou escrita) através do sistema de controle, gerenciadas por um usuário (operador do sistema), têm pacotes de dados da ordem de algumas dezenas de bytes. Portanto, a taxa de transmissão não é um parâmetro crítico do sistema, descartando a necessidade de meios de comunicação que ofereçam alta velocidade de transmissão[1], os quais, em geral, apresentam maior custo e complexidade de integração.

Considerou-se o protocolo CAN, definido pelo padrão ISO-11898, como o mais adequado para atender os requisitos do sistema. Esta escolha deu-se em função das seguintes características: i) a camada física utiliza transmissão diferencial através de um par trançado, ii) o acesso ao barramento é arbitrado através da técnica *non-destructive bit-wise*, iii) as mensagens são pequenas (no máximo 8 bytes de dados) e protegidas por *checksum*, iv) cada mensagem carrega um valor numérico que define a sua prioridade no barramento e também funciona como um identificador, v) o *hardware* tem um gerenciamento elaborado de erros, o que resulta na retransmissão de mensagens que não foram recebidas corretamente, vi) existem meios efetivos para isolar nós defeituosos no barramento. O padrão ISO-11898 define os requisitos elétricos que um transceptor CAN deve atender, tais como: a faixa de tensão de modo comum no barramento, a tensão diferencial no barramento, a resistência interna, a capacitância interna, etc. Além disso, o padrão define a taxa de transferência máxima em 1 Mbps, onde uma linha de transmissão pode ter até 40 metros de comprimento. Linhas de transmissão mais longas podem ser utilizadas se for reduzida a taxa de transferência. Comparando o CAN ao *Ethernet*, que atenderia principalmente ao requisito de rede do sistema (*daisy chain*), o primeiro mostra-se mais adequado à taxa de transmissão requerida e possui menor complexidade de integração. Caso o *Ethernet* fosse escolhido, haveria aumento tanto no custo quanto no tempo de desenvolvimento do projeto. Meios de comunicacao como o RS-232 ou USB não atenderiam os requisitos do sistema por serem projetados para operações em curta distância e não suportarem operação em rede (*daisy chain*). A quantidade de circuitos de controle no barramento será limitada pelo transceptor CAN escolhido. Este será apresentado na próxima seção em conjunto com a descrição do protótipo desenvolvido para o circuito de controle e o detalhamento das características de um barramento CAN construído para testes.

3. O PROTÓTIPO DO CIRCUITO DE CONTROLE

O circuito de controle que permite a configuração e a depuração do módulo de aquisição e digitalização de sinais está representado na Figura 3 através de um diagrama em blocos que ilustra os principais componentes deste circuito.

[1] Um exemplo é o *Ethernet*, que atualmente oferece taxa de transmissão de 1 Gbps.

Estes componentes são o microcontrolador – PIC18F2680 e o *transceiver* CAN – MCP2551, suas funções serão apresentadas a seguir. Para verificar as funcionalidades do referido circuito de controle, um protótipo foi projetado para ser construído em paralelo com o desenvolvimento dos demais circuitos que compõem o NDAQ. O protótipo integra

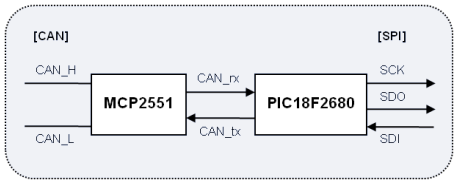


Figura 3: Diagrama em Blocos do Circuito de Controle.

os dois componentes já mencionados do circuito de controle e também outros três principais componentes, como ilustra o diagrama em blocos da Figura 4. O circuito impresso do protótipo foi projetado na ferramenta *Altium Designer* [7] e confeccionado por uma máquina de prototipagem rápida instalada no CBPF. Uma fotografia desta placa é mostrada na Figura 5. A função dos principais componentes eletrônicos do protótipo será descrita abaixo. Componentes que não estão listados formam a fonte de alimentação do circuito e podem ser vistos no esquema elétrico completo, ver Apêndice B desta nota técnica.

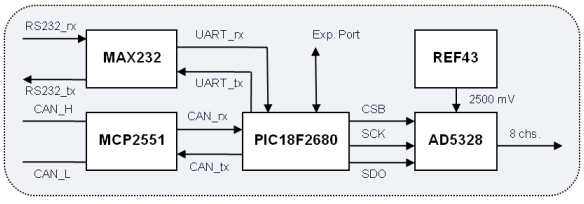


Figura 4: Diagrama em Blocos do Protótipo do Circuito de Controle.

1. **Microcontrolador PIC18F2680** [8]: Um microcontrolador é um dispositivo que normalmente integra uma unidade central de processamento – CPU, memória RAM, memória de programa e diversos periféricos. O PIC18F2680 possui dois periféricos indispensáveis para o circuito de controle apresentado: o controlador de comunicação CAN e o controlador de comunicação SPI. O programa executado no microcontrolador tem a função de gerenciar as mensagens que trafegam pelo barramento CAN, executando operações de escrita ou de leitura através das linhas SPI. Cada mensagem CAN possui um identificador para endereçar a operação de leitura ou escrita com um determinado registrador de um dos dispositivos conectados ao barramento de comunicação SPI. Este identificador será decodificado pelo programa do microcontrolador e o devido dispositivo que se quer controlar será acessado. Portanto, uma mensagem CAN necessariamente endereça um dos dispositivos controlados pela porta SPI e também endereça um registrador deste dispositivo, estes serão referidos como par dispositivo-registrador neste documento. No caso de uma operação de escrita, esta mesma mensagem CAN transporta o dado que se deseja escrever no dispositivo-registrador. Já a operação de leitura, começa com uma mensagem CAN que apenas endereça o dispositivo-registrador e o dado lido volta como uma mensagem CAN de mesmo endereço para o requisitante.

2. ***Transceptor* MCP2551** [9]: Converte o padrão elétrico do controlador CAN, embutido no microcontrolador, no padrão elétrico para um barramento CAN. Este componente faz esta conversão de padrão elétrico para ambos os sentidos da comunicação e atende os requisitos do padrão ISO-11898. O transceptor permite um total de 112 nós em um barramento CAN. Para o sistema de controle do NDAQ, um dos nós é o computador que executa a aplicação de controle. Portanto, pode-se concluir que 111 circuitos de controle podem ser acessados.

3. **Conversor digital-analógico AD5328** [10]: É um componente capaz de gerar oito canais simultâneos e independentes de tensão analógica, com valores entre 0 e uma tensão de referência, que no caso do circuito de controle é de 2,5 Volts. A configuração deste componente é feita através da escrita em registradores através de comunicação SPI. Este componente também é utilizado no NDAQ e foi inserido no protótipo para que se possa verificar a comunicação com um dos componentes que realmente será utilizado no módulo.

4. **Gerador de Tensão REF43**: Este componente gera uma tensão de 2,5 Volts para ser utilizada como referência pelo conversor digital-analógico AD5328.

5. ***Transceiver* RS-232**: Como o microcontrolador escolhido possui também um periférico UART, acrescentou-se no protótipo um *transceiver* RS-232. Tal componente traduz, em ambos os sentidos da comunicação, o padrão elétrico do controlador UART no padrão elétrico RS-232, que é um protocolo de comunicação serial assíncrono. Esta porta de comunicação é normalmente encontrada em computadores pessoais onde pode-se desenvolver o programa para o microcontrolador, através de ferramentas específicas que serão apresentadas no seguinte tópico. Assim, a comunicação RS-232 é um meio de se transferir o programa desenvolvido em um computador pessoal para o microcontrolador no protótipo, com o auxílio de um *bootloader* previamente programado no mesmo. Com esta prática elimina-se a necessidade de se utilizar um dispositivo programador específico para o microcontrolador, exceto para a programação inicial do *bootloader*. Detalhes sobre o *bootloader* e sobre a sua utilização estão disponíveis no Apêndice A deste documento.

O barramento CAN que foi construído para os testes do protótipo do circuito de controle, que serão apresentados nos dois tópicos seguintes, está esquematizado na Figura 6. O nó 1 do barramento é o protótipo do circuito de controle, enquanto o nó 2 é um dispositivo comercial – Kvaser Leaf

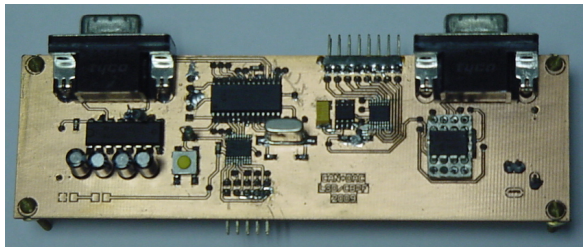


Figura 5: Placa de Circuito Impresso do Protótipo do Circuito de Controle.

HS [11] – que funciona como uma interface CAN – USB, ver Figura 7. O nó 1 do barramento CAN foi ligado a

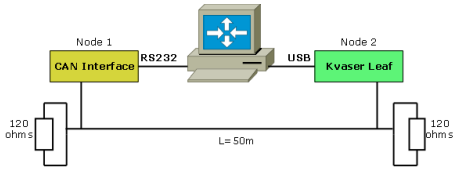


Figura 6: Barramento CAN utilizado.

um computador pessoal através da porta RS-232 disponível no protótipo, tanto para programação do microcontrolador quanto para a transferência de dados específicos da aplicação que será apresentada. Já o nó 2, foi ligado ao mesmo computador pessoal através da porta USB do Kvaser Leaf HS. Ambos os nós do barramento foram configurados para transferir dados a 125 Kbps, utilizando o modo de 11 bits para identificação das mensagens CAN. O meio físico deste barramento é um par trançado de fios de cobre (sem blindagem), com 50 metros de comprimento e terminado com resistores de 120 ohms nas extremidades. A terminação é um requisito do padrão ISO-11898.

Figura 7: Interface CAN – USB comercial – Kvaser Leaf HS.

4. TESTES DE COMUNICAÇÃO ATRAVÉS DO BARRAMENTO CAN

Nesta seção serão apresentados dois testes que foram realizados com o propósito de verificar o funcionamento do protótipo do circuito de controle e a integridade do barramento CAN, ambos descritos na seção anterior. Os dois dispositivos conectados ao barramento CAN, o protótipo do circuito de controle e a interface CAN – USB (Kvaser Leaf HS) serão respectivamente referidos, ao logo do texto, como nó 1 e nó 2.

O primeiro teste realizado baseia-se em um programa para o microcontrolador do nó 1, para que este se comporte como uma interface RS-232 – CAN. Neste teste, o nó 1 recebe dados através da porta RS-232 e os encapsula em mensagens CAN. Estas mensagens são entregues ao barramento CAN e chegam ao nó 2, onde os dados enviados são reproduzidos. O sentido contrário da comunicação também é possível, onde dados enviados pelo nó 2 são reproduzidos pelo nó 1. O computador pessoal utilizado no teste executa duas aplicações simultaneamente: TeraTerm [12] – aplicação emuladora de terminal[2], que se comunica com o nó 1, enviando e recebendo dados pela porta RS-232 e CANKing [13] – aplicação para monitorar e depurar barramentos CAN, que controla o nó 2.

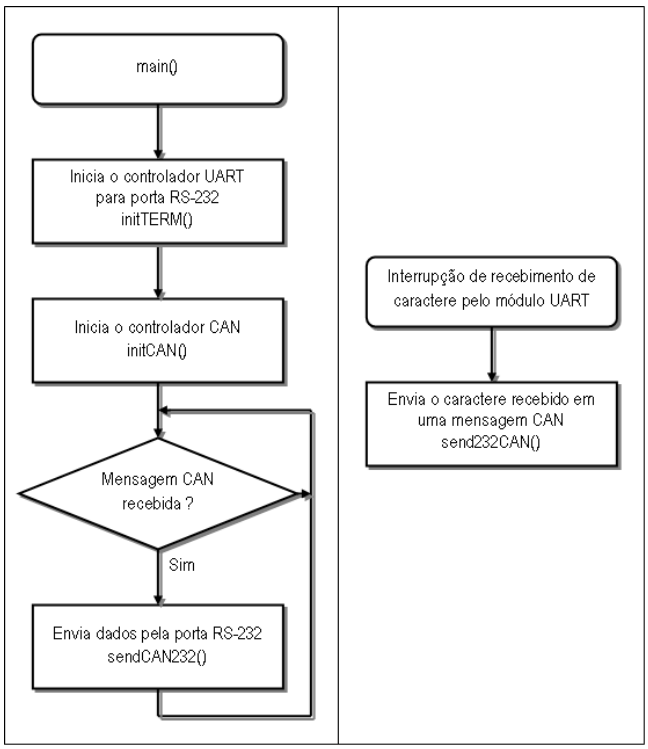


Figura 8: Fluxograma da rotina principal do programa para o microcontrolador no primeiro teste.

A Figura 8 mostra os fluxogramas da rotina principal do programa executado no microcontrolador e do tratamento da interrupção gerada pela recepção de um caractere através da porta RS-232. A rotina principal configura o controlador da comunicação RS-232, em seguida o controlador CAN, e por último fica em um *loop* infinito, onde toda vez que uma mensagem CAN for recebida, esta terá seus 8 bytes de dados enviados para a porta RS-232. Pode-se concluir que a rotina principal maneja os dados que chegam ao nó 1 pelo barramento CAN e os envia para a porta RS-232. Já os dados que chegam ao nó 1 pela porta RS-232, geram uma interrupção

[2] Qualquer outra aplicação emuladora de terminal, capaz de controlar uma porta RS-232 de um computador pessoal, pode ser utilizada.

no microcontrolador que os envia para o barramento CAN através da função send232CAN(). Os fluxogramas da rotina principal e do tratamento da interrupção fazem referência a sub-rotinas que são mostradas nas Figuras 9 e 10.

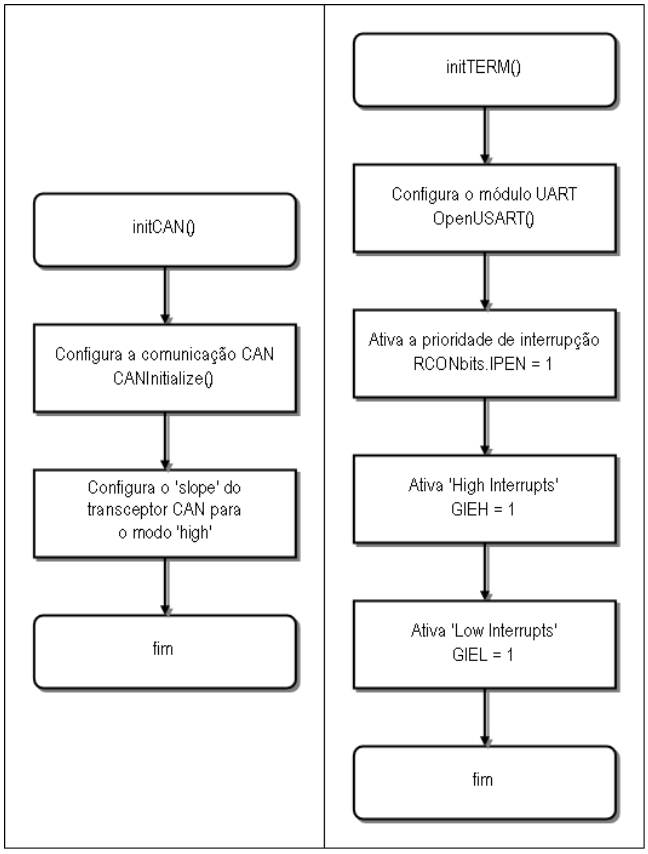


Figura 9: Fluxogramas das sub-rotinas que configuram o controlador CAN e o controlador UART.

Para o segundo teste realizado, outro o programa foi desenvolvido para o microcontrolador do nó 1. Com este programa, o nó 1 monitora todas as mensagens lançadas no barramento CAN e retorna para o mesmo apenas as mensagens que possuem endereço '1'. O nó 2 continua sendo controlado pela aplicação CANKing, que é configurado para executar dois processos: gerar mensagens de conteúdo aleatório com endereço '1' e colocá-las no barramento e monitorar o mesmo por mensagens com endereço 1. Desta maneira, toda mensagem gerada pelo nó 2 deverá ser repetida pelo nó 1 e chegar de volta ao nó 2. A quantidade de mensagens geradas e a de mensagens devolvidas é contada no CANKing, para que a razão entre estes valores determine a integridade do tráfego das mensagens no barramento.

A Figura 11 ilustra o fluxograma da rotina principal do programa do segundo teste que é executado no microcontrolador. Esta rotina inicia o controlador CAN e depois mantém-se em um *loop* infinito, onde toda mensagem CAN recebida é tratada pela sub-rotina loopbackCAN(), ilustrada na Figura 11. Nesta sub-rotina, compara-se o endereço da mensagem CAN com o valor '1' e se o resultado for verdadeiro, os dados desta mensagem são colocados no barramento CAN através de outra mensagem com o mesmo endereço.

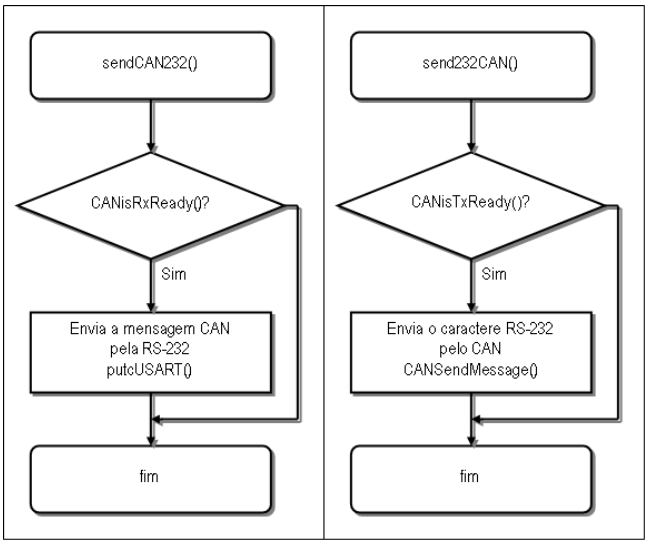


Figura 10: Fluxogramas da sub-rotina que envia dados recebidos pelo barramento CAN para a porta RS-232 e da sub-rotina que implementa o sentido contrário da comunicação (direita).

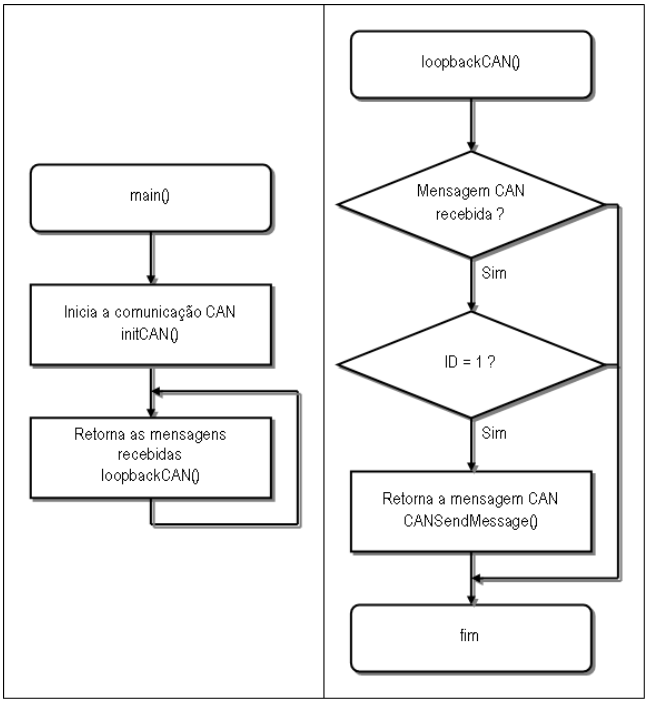


Figura 11: Fluxogramas da rotina principal e da sub-rotina utilizada para o microcontrolador no segundo teste.

Os programas para o microcontrolador, em ambos os testes, foram desenvolvidos utilizando a linguagem de programação C. Utilizou-se a versão para uso acadêmico gratuita do compilador de linguagem C MPLAB C18 [14], embutido no ambiente de desenvolvimento integrado MPLAB [15], ambos desenvolvidos pela fabricante do microcontrolador – Microchip. As funções CANisRxReady(), CANisTxReady() e CANSendMessage() fazem parte da biblioteca de funções para o controlador CAN – PIC18C CAN Routines in 'C' [16], também desenvolvida pela Microchip.

A função putcUSART(), definida no cabeçalho 'usart.h', faz parte da biblioteca de funções para controle de periféricos do microcontrolador, que é parte do compilador C18.

A Figura 12 mostra uma captura de tela do computador utilizado durante o primeiro teste descrito. Nesta imagem, pode-se destacar a janela da aplicação CANKing, onde a configuração do nó 2 foi feita na janela 'CAN 1', e a janela da aplicação Tera Term, conectada a porta RS-232 do nó 1, através da porta 'COM 1' do computador. No terminal emulado Tera Term é possível entrar com caracteres e também recebê-los. Os caracteres 'P', 'r', 'o', 'j', 'e', 't', 'o', assim como os caracteres 'A', 'n', 'g', 'r', 'a', foram digitados no terminal e chegaram ao nó 1 através da comunicação RS-232, onde foram encapsulados em mensagens CAN e enviados para o nó 2. É possível checar o recebimento destes caracteres através da janela 'Output Window' do CANKing. O conjunto de caracteres 'Neutrino' foi digitado na 'String Page'[3] do CanKing, enviado do nó 2 para o nó 1, onde o conjunto de caracteres foi decodificado da mensagem CAN e cada caractere foi enviado, através da porta RS-232, ao terminal.

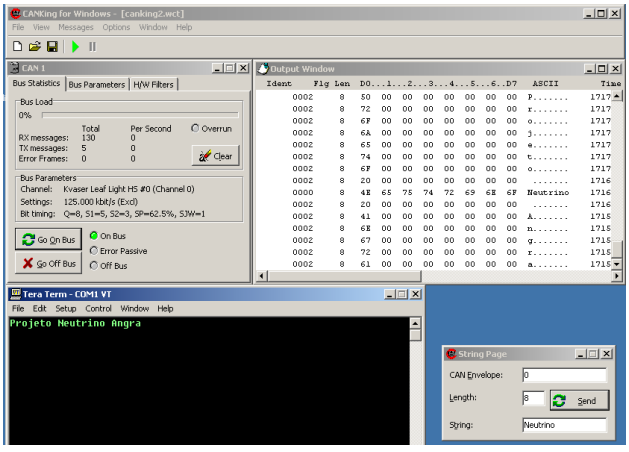


Figura 12: Captura de Tela do Primeiro Teste.

No segundo teste do protótipo, pode-se destacar a opção 'CAN Bus Loader' do aplicativo CANKing, onde é possível gerar mensagens CAN de conteúdo aleatório ou definido, configurar identificadores ou equivalentemente endereços, configurar a taxa de mensagens e etc. A Figura 13 mostra uma captura de tela do computador durante o teste. Em cerca de 50 minutos, 4.159.597 mensagens CAN foram enviadas pelo nó 2 e retornadas pelo nó 1, o que demonstrou integridade total do barramento CAN construído.

5. CONTROLANDO UM DAC ATRAVÉS DO BARRAMENTO CAN

Nesta seção será apresentada uma aplicação que permite o controle de um conversor digital-analógico – DAC – presente no protótipo do circuito de controle, descrito na seção 3 desta

[3] Opção para enviar strings do CANKing, disponível no menu 'Messages'.

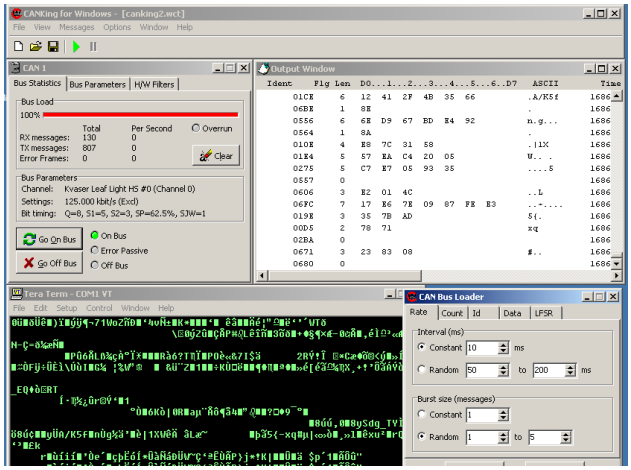


Figura 13: Captura de Tela do Segundo Teste.

nota técnica, a partir de um computador através de um barramento CAN. Este computador se conecta ao barramento CAN através de uma interface CAN - USB (Kvaser Leaf HS) e executa um programa dedicado ao controle do referido DAC. O protótipo do circuito de controle e o conjunto formado pelo computador pessoal e a interface CAN - USB serão respectivamente referidos como nó 1 e nó 2 ao longo do texto. O barramento CAN utilizado para comunicação destes nós também está descrito na seção 3 deste documento.

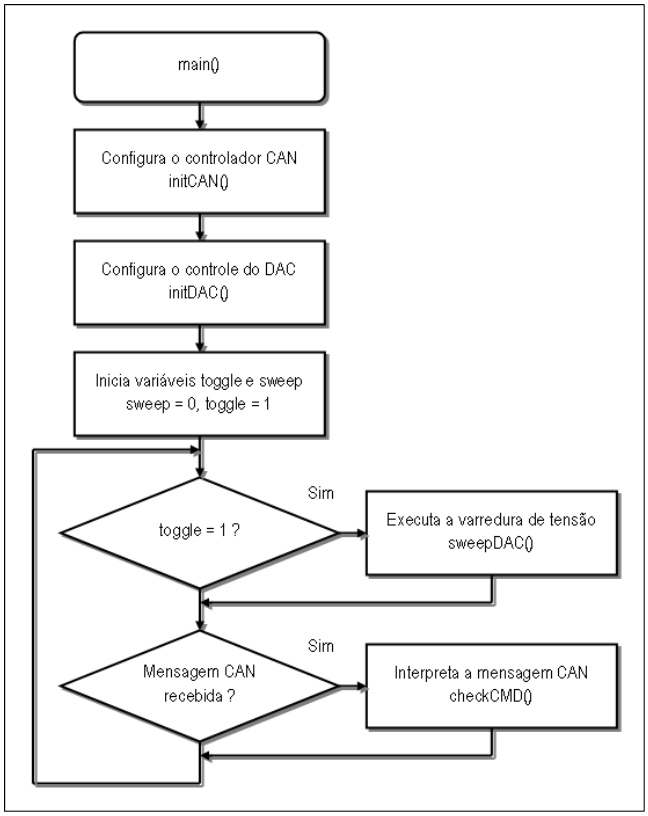


Figura 14: Fluxograma da rotina principal do programa para o microcontrolador.

Para que se possa controlar o DAC mencionado a partir do barramento CAN, o microcontrolador do nó 1 deve se com-

portar como uma ponte entre o controlador CAN e o controlador SPI, sendo o primeiro responsável pela comunicação com o barramento CAN e o segundo pela comunicação com o DAC. Deste modo, o programa desenvolvido para o microcontrolador interpreta as mensagens CAN recebidas e caso esta contenha um comando para o DAC, a operação desejada será executada no mesmo. A Figura 14 mostra o fluxograma da rotina principal deste programa. Esta rotina configura o controlador CAN, o controlador SPI, parâmetros iniciais do DAC e por último mantém-se em um elo infinito executando as sub-rotinas de interpretação das mensagens CAN e de execução de comandos para o DAC. Os fluxogramas destas sub-rotinas podem ser vistos nas Figuras 9, 15, 16 e 17. As sub-rotinas que configuram os periféricos de controle CAN e SPI estão descritas abaixo, para que os parâmetros de configuração sejam detalhados.

- **initCAN()**: Esta sub-rotina configura o controlador CAN embutido no microcontrolador através da função CANInitialize(), que faz parte da biblioteca PIC18C CAN Routines in 'C' [16]. Os parâmetros definidos são a taxa de transferência, configurada em 125 Kbps e o modo de operação do nó, configurado para receber qualquer mensagem do barramento. Em seguida, a porta RB4 do microcontrolador é colocada em nível baixo, aproximadamente zero Volts, para manter o transceptor CAN operando no modo de transição de sinal rápido – *high slew rate mode*. Mais informações podem ser consultadas na folha de dados do microcontrolador [8], nas notas de aplicação 738 [16] e 713 [17] e na folha de dados do transceptor CAN [9]. O fluxograma desta sub-rotina está disponível na Figura 9.

- **initDAC()**: Nesta sub-rotina, o controlador SPI do microcontrolador é configurado, como mostra a Figura 15, através da função OpenSPI() que acompanha a biblioteca de funções para periféricos do compilador Microchip C18 [14]. Os parâmetros configurados são: taxa de transferência SPI, definida como um quarto da frequência do *clock* principal e modo de operação, definido como modo 0,0. Detalhes sobre a configuração do controlador SPI estão disponíveis da folha de dados do microcontrolador, na seção 17.0 – Master Synchronous Serial Port (MSSP) Module. Neste ponto, a sub-rotina começa a comunicação com o DAC, ativando o sinal de seleção do mesmo – *chip select* – e escreve uma palavra de configuração inicial, definida com base na folha de dados do DAC [10]. Por último, a comunicação com o DAC é encerrada ao desligar-se o sinal de seleção.

O programa dedicado ao controle do DAC, executado no computador do nó 2, é responsável por enviar as mensagens de controle para o barramento CAN de acordo com o que foi definido na interpretação de comandos do nó 1. Tal programa de controle denomina-se Dacc, uma abreviação de DAC e controle, e é uma aplicação em modo texto que utiliza argumentos da linha de comando para a sua operação. Estes argumentos estão listados abaixo.

- **dacc.exe -t**: Inverte o estado de ligado para desligado ou vice-versa da função de varredura de tensão para o DAC.

- **dacc.exe -v [valor]**: Configura um valor de tensão constante, através do parâmetro [valor], para ser gerado pelo DAC. Como o DAC utilizado é de 12 bits, o parâmetro [valor] pode variar de 0 a 4095 em passos unitários, onde 0 define a menor tensão gerada – 0 Volts – e 4095 a máxima – 2,5 Volts. O passo unitário corresponde à menor variação de tensão possível, dada pelo quociente entre a tensão máxima e o valor 4096, ou seja, 610,35 uV.

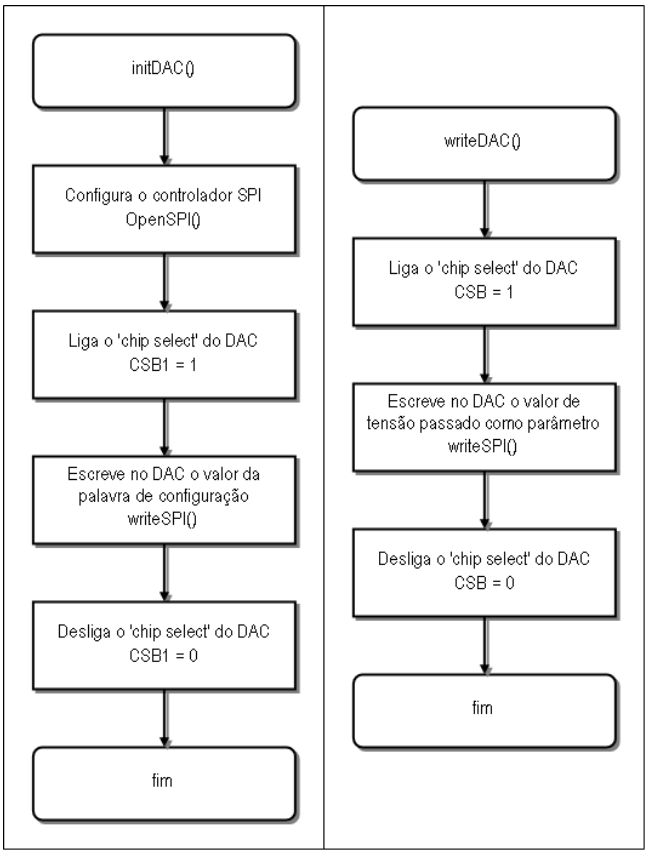


Figura 15: Fluxogramas das sub-rotinas que configura o DAC (esquerda) e escreve no mesmo.

O programa Dacc utiliza a biblioteca de funções CANLib [18] para acessar a interface CAN - USB do nó 2 e foi compilado para o sistema operacional Windows utilizando o Microsoft Visual C++ 2008 Express Edition[4]. Existem versões da biblioteca CANLib para o sistema operacional Linux, o que possibilita a compilação do Dacc neste sistema operacional, com algumas modificações. A Figura 18 mostra o fluxograma da rotina principal do programa Dacc, onde o controlador CAN da interface CAN - USB é configurado, os argumentos da linha de comando são interpretados e a

[4] Express Edition é a versão gratuita da ferramenta de desenvolvimento, que possui algumas limitações dispensáveis para a aplicação descrita nesta nota técnica.

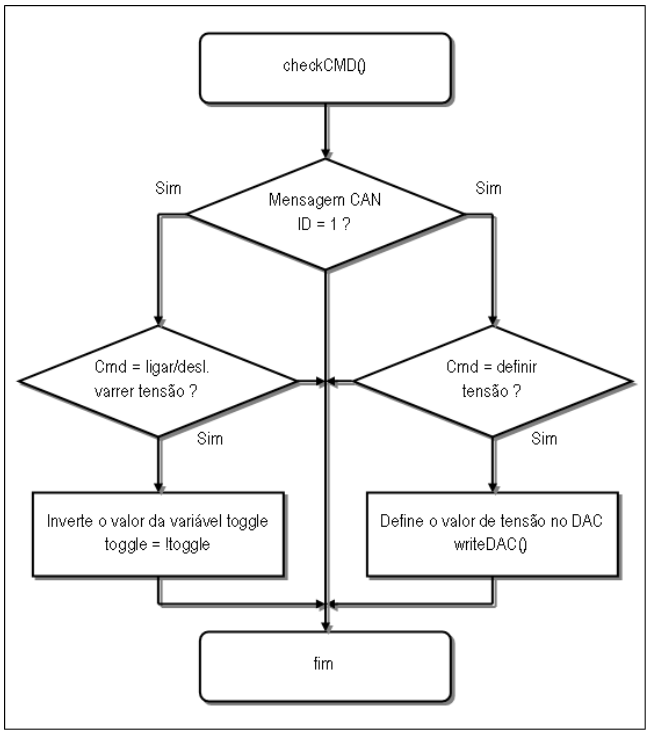


Figura 16: Fluxograma da sub-rotina do programa para o microcontrolador que interpreta os comandos recebidos pelo barramento CAN.

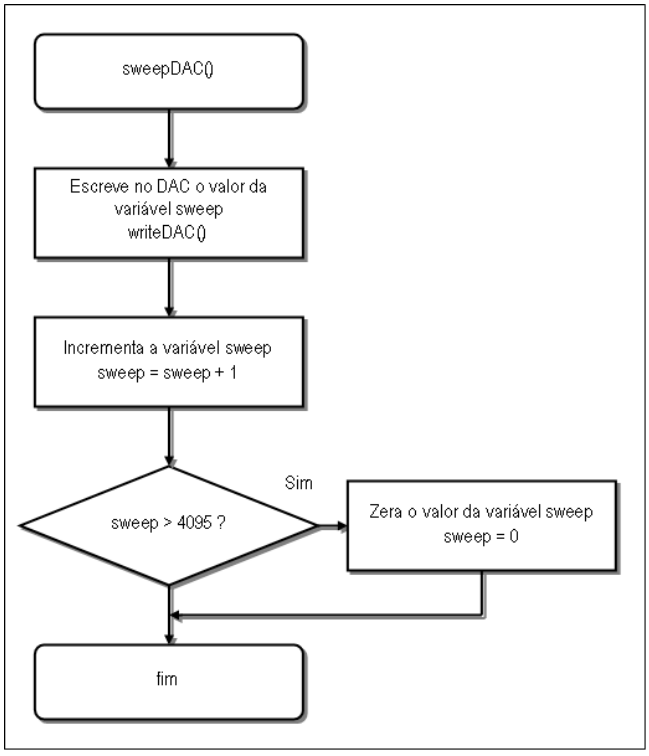


Figura 17: Fluxograma da sub-rotina do programa para o microcontrolador que gera a varredura de tensão, ou rampa de tensão.

mensagem CAN é enviada. A sub-rotina que configura a comunicação CAN está representada por um fluxograma na mesma figura.

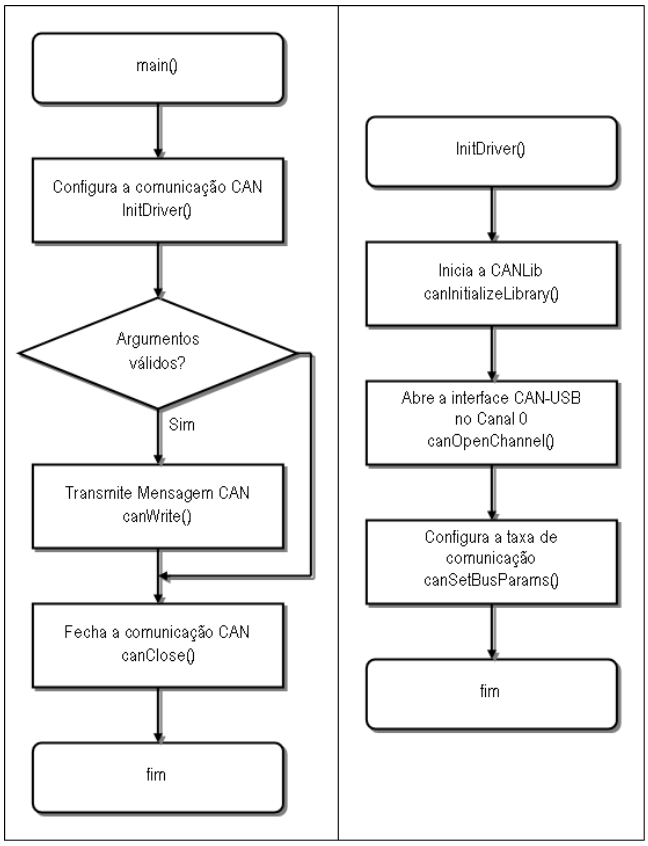


Figura 18: Fluxogramas da rotina principal da aplicação desenvolvida para um computador pessoal e da sub-rotina utilizada.

Para mais informações sobre as funções canInitializeLibrary(), canOpenChannel(), canSetBusParams, canWrite() e canClose(), assim como a utilização da CANLib, a documentação que acompanha a mesma pode ser consultada.

Com a função de varredura de tensão ligada, foram adquiridas as formas de onda do canal 1 e do canal 8 do DAC. A Figura 19 mostra uma captura de tela do osciloscópio durante a aquisição de sinais, onde o canal 8 é a forma de onda inferior da imagem. Ambos os canais variam de aproximadamente 0 até 2,5 Volts, o que demonstra que o DAC está excursionando pelos 4096 valores de tensão possíveis. Uma visão mais detalhada da rampa de tensão do canal 1 está disponível na Figura 20. Como o DAC utilizado atualiza paralelamente todos os seus canais não há diferença de tempo entre o início da rampa de tensão do canal 1 e a do canal 8, apesar da sub-rotina sweepDAC() escrever a configuração para cada canal do DAC sequencialmente. A varredura de todas as tensões possíveis no DAC dura cerca de 1 segundo, como mostra a Figura 21. Como a sub-rotina sweepDAC() atualiza todos os canais do DAC, pode-se concluir que o microcontrolador configura 32768 valores de tensão por segundo no DAC. Este valor é o produto entre os 4096 valores de tensão possíveis e os 8 canais do DAC.

O alinhamento entre as rampas de tensão do canal 1 e do canal 8 pode ser observado na Figura 22, que também é uma captura de tela do osciloscópio durante o processo de aquisição de sinais descrito anteriormente, porém com uma escala de tempo menor.

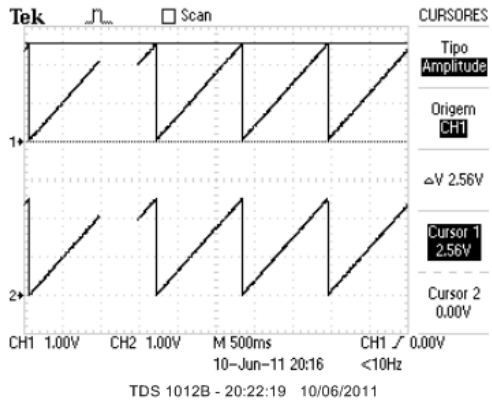


Figura 19: Captura de tela do osciloscópio mostrando as rampas de tensão geradas pelos canal 8 e canal 1 (superior) do DAC.

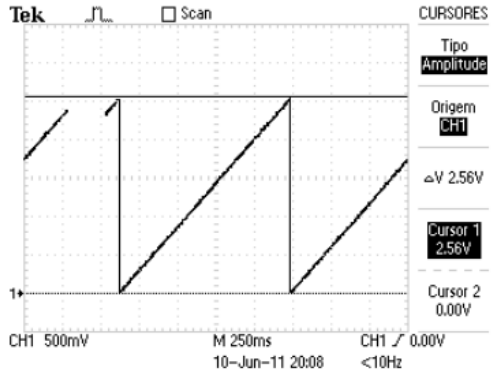


Figura 20: Captura de tela do osciloscópio mostrando a amplitude máxima da rampa de tensão gerada pelo canal 1 do DAC.

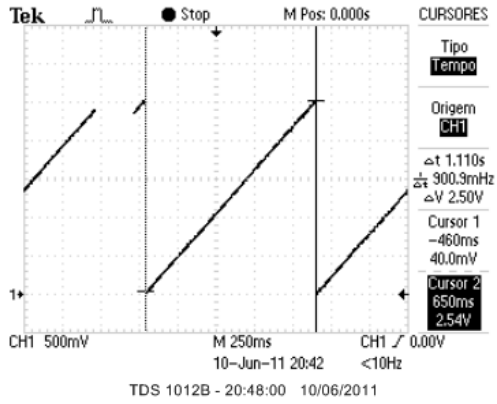


Figura 21: Captura de tela do osciloscópio mostrando o período da rampa de tensão gerada pelo canal 1 do DAC.

Utilizando a função de tensão constante para controlar o DAC, seis valores de tensão foram gerados variando de 0 a 2,5 Volts, em passos de 0,5 Volts. Para cada valor enviado, 10.000 amostras da tensão gerada nos canais do DAC foram medidas. A aquisição destes sinais foi feita através de um osciloscópio Tektronix 1012B ligado à porta USB de um computador pessoal. A comunicação entre o osciloscópio e

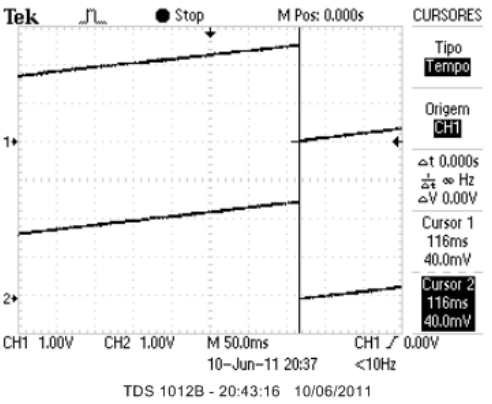


Figura 22: Captura de tela do osciloscópio mostrando o alinhamento entre as rampas de tensão geradas pelo DAC.

o computador foi estabelecida através do aplicativo OpenChoice Desktop [19], que trabalha em conjunto com a ferramenta TekVISA, estabelecendo o protocolo VISA para comunicação. A Figura 23 mostra os histogramas dos valores adquiridos para cada tensão enviada e as Figuras 24

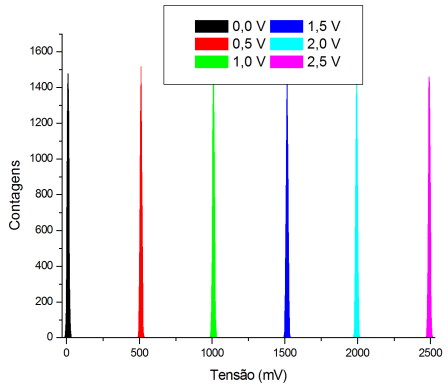


Figura 23: Histogramas das tensões geradas pelo DAC. A legenda mostra os valores de tensão comandados.

e 25 mostram os histogramas das tensões de 0 e 2,5 Volts separadamente para uma visão detalhada da distribuição do ruído aleatório na tensão gerada, a largura das gaussianas a meia altura é de aproximadamente 13 mV. Um ajuste linear dos valores médios das distribuições gaussianas de cada histograma é mostrado na Figura 26. As barras de erro, derivados da distribuição do ruído aleatório na medição, foram multiplicadas por 10 para permitir melhor visualização. A reta demonstra a linearidade do canal 1 do DAC.

6. CONCLUSÃO

Um protótipo de circuito de controle baseado em barramento CAN foi construído e testado. Este circuito está incorporado a um módulo digitalizador do Projeto Neutrinos Angra. Em uma primeira aplicação, apresentada neste documento, o protótipo foi utilizado para verificar o comportamento do meio de comunicação escolhido - o barra-

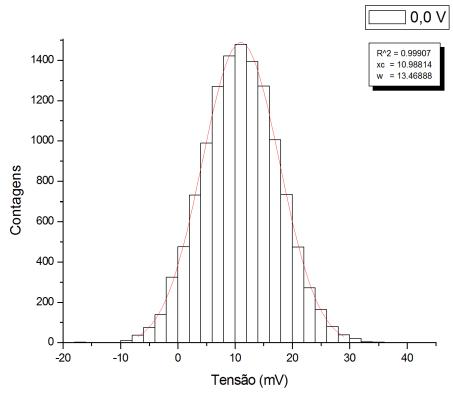


Figura 24: Histograma para a tensão comandada de 0,0 Volts.

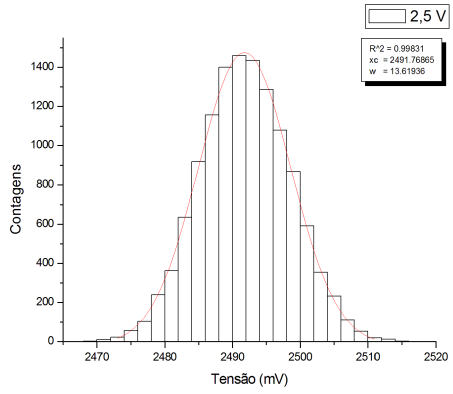


Figura 25: Histograma para a tensão comandada de 2,5 Volts.

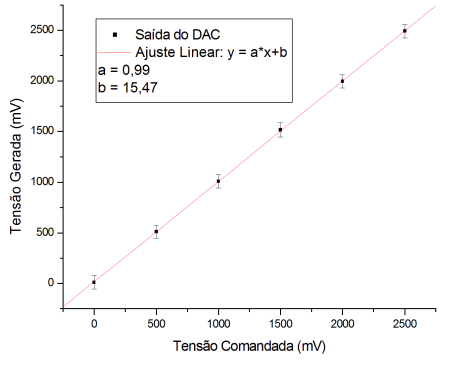


Figura 26: Ajuste linear dos valores de tensão gerados pelo DAC.

mento CAN. Dois testes foram executados nesta aplicação, de forma que os resultados finais comprovaram que o meio de comunicação é capaz de atender as necessidades do circuito de controle. Em uma segunda aplicação, um conversor digital-analógico (DAC), incluído no protótipo, foi controlado através de um computador pessoal. Este computador, por sua vez, estava conectado ao barramento CAN através de uma interface CAN-USB. O DAC é um dos dispositivos eletrônicos do módulo digitalizador que será controlado remotamente através do barramento CAN. Nesta segunda aplicação foi possível alterar a tensão de saída de todos os canais do DAC, verificando-se inclusive o sincronismo entre canais.

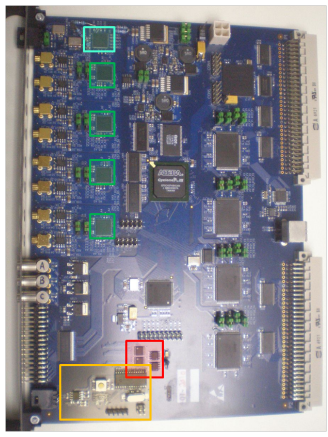


Figura 27: Módulo eletrônico digitalizador - NDAQ - onde está inserido o circuito de controle.

A versão final do circuito de controle, já inserida no módulo digitalizador, é similar ao protótipo apresentado, excluindo-se somente a parte de comunicação RS-232. O programa do microcontrolador, desenvolvido para o controle do conversor digital-analógico, foi utilizado como base para um programa mais completo, capaz de controlar todos dispositivos eletrônicos do módulo digitalizador. A figura 27 mostra uma foto do módulo digitalizador, com destaque para a região ocupada pelo circuito de controle e para os dispositivos que são controlados: em laranja, o circuito de controle; em vermelho, dois conversores digital-analógico e circuitos integrados auxiliares; em verde, quatro conversores analógico-digital; e em azul, um circuito integrado distribuidor de sinal de *clock*.

Apêndice A: *BOOTLOADER* PARA O MICROCONTROLADOR

O *bootloader* utilizado no microcontrolador é um programa residente numa área da memória de programa especial. Sua finalidade é permitir que um código de programa seja transferido para o microcontrolador, a partir de um computador pessoal, sem a necessidade de uma ferramenta dedicada externa, que normalmente representa um custo a mais no projeto. Ao aproveitar o controlador UART embutido no microcontrolador, o *bootloader* reduz o circuito de programação a um único circuito integrado transceptor RS-232. Desta maneira, o computador pessoal executa uma aplicação de programação do microcontrolador que se comunica com o *bootloader* através da porta RS-232. O *bootloader* adotado foi o *Colt bootloader*, desenvolvido por M. Dubuc com base na versão do fabricante do microcontrolador, descrita em [20]. Mais detalhes sobre esse *bootloader* podem ser vistos em [21]. A seguir são detalhados quatro itens necessários para a utilização desse *bootloader* com o PIC18F2680, microcontrolador utilizado nas aplicações desta nota técnica.

- **Obtenção e programação do binário do *bootloader* para o PIC18F2680**

  O arquivo binário do *bootloader* que deve ser utilizado é o 'bootload2k.hex', disponível em [21]. Esse arquivo deve ser programado no microcontrolador através de uma ferramenta dedicada. A ferramenta utilizada foi o programador PICkit 3, mas qualquer outra similar pode ser empregada. Uma vez que o *bootloader* esteja residente na memória do microcontrolador, este será capaz de receber programas quantas vezes forem necessárias, através do *bootloader*. Antes da transferência do *bootloader*, os bits de configuração do microcontrolador precisam ser configurados. Estes bits definem parâmetros como o tipo do oscilador utilizado, estado dos temporizadores, configurações de portas, etc. Estes bits devem possuir configurações específicas para cada projeto. Para os projetos apresentados neste documento, os bits de configuração estão mostrados na Tabela 1.

- **Mudança do *entry point* nos binários de inicialização para o MCC18**

  O compilador de linguagem C - Microchip C18 - utiliza códigos pré-compilados para iniciar o microcontrolador. Estes códigos acompanham o compilador e dependem do endereço de memória onde começa o programa que deve ser executado. Como o *bootloader* utilizado ocupa o início da memória de programa, o endereço do início do programa que pretende-se executar – *entry point* – deve ser realocado, levando-se em consideração a área de memória ocupada pelo *bootloader*. O processo descrito a seguir funcionará somente para o referido compilador.

  1. Faça cópias dos códigos de inicialização pré-compilados originais. No diretório onde o compilador foi instalado, localize o subdiretório [\lib] e dentro deste, renomeie os arquivos [c018.o], [c018i.o] e [c018iz.o] para [c018_copia.o], [c018i_copia.o], [c018iz_copia.o].
  2. No diretório onde o compilador foi instalado, localize o subdiretório [\src\traditional\startup]. Edite os arquivos c018.c, c018i.c e c018iz.c, procure pela diretiva [#pragma code _entry_scn=0x000000] e a altere para [#pragma code _entry_scn=0x000800]. Este procedimento altera o *entry point* para o endereço 0x800, sendo este valor o tamanho ocupado pelo *bootloader* na memória.
  3. No diretório onde o compilador foi instalado, localize o subdiretório [\src] e execute o arquivo [make_genericlibs_t.bat], o que irá compilar os códigos de inicialização.
  4. Volte ao diretório [\lib] e renomeie os arquivos que acabaram de ser gerados na compilação, [c018.o], [c018i.o] e [c018iz.o] para [c018_800.o], [c018i_800.o] e [c018iz_800.o]. Esses arquivos são os códigos de inicialização pré-compilados que funcionarão com o *bootloader*.
  5. Para finalizar, ainda no diretório [\lib], retorne os arquivos renomeados no primeiro passo para os nomes originais. Desta maneira, esses arquivos poderão ser usados em compilações que não utilizam o *bootloader*.

- **Configuração do arquivo que auxilia o *linker* do compilador.**

  O compilador mencionado no item anterior utiliza um programa montador de arquivos binários, conhecido como *linker*. Esse programa agrega os diversos objetos binários gerados no processo de compilação em um único binário final. Tal processo é baseado em referências de endereços de memória do dispositivo onde pretende-se executar o programa. Assim, o *linker* funciona baseado em um arquivo de configuração, conhecido como *linker script*, que deve estar de acordo com o mapa de memória do microcontrolador do projeto. O *linker script* utilizado na compilação deve ser alterado para levar em consideração a área de memória ocupada pelo *bootloader*. Cada projeto possui um *linker script* específico, de acordo com as necessidades do mesmo. O arquivo utilizado para os projetos apresentados nesta nota técnica está disponível na Figura 28 e pode ser usado como exemplo. Repare que o código de inicialização do microcontrolador declarado no arquivo é o [c018i_800.o], citado no item anterior.

- **Obtenção do aplicativo de programação do microcontrolador a partir de um computador pessoal.**

  O aplicativo que deve ser executado em um computador pessoal para transferir um programa para o microcontrolador faz parte do *Colt bootloader*, e está disponível em [21]. Este aplicativo possui uma interface gráfica com o usuário onde pode-se escolher o programa que se deseja transferir. Caso um programa nunca tenha sido transferido para o microcontrolador, quando o último for energizado ou reiniciado, o *bootloader* irá esperar indefinidamente pela transferência. No caso em que já exista um programa na memória, o *bootloader* irá esperar por 2 segundos para que o usuário transfira um novo programa. Depois desses 2 segundos o programa que já reside na memória será executado.

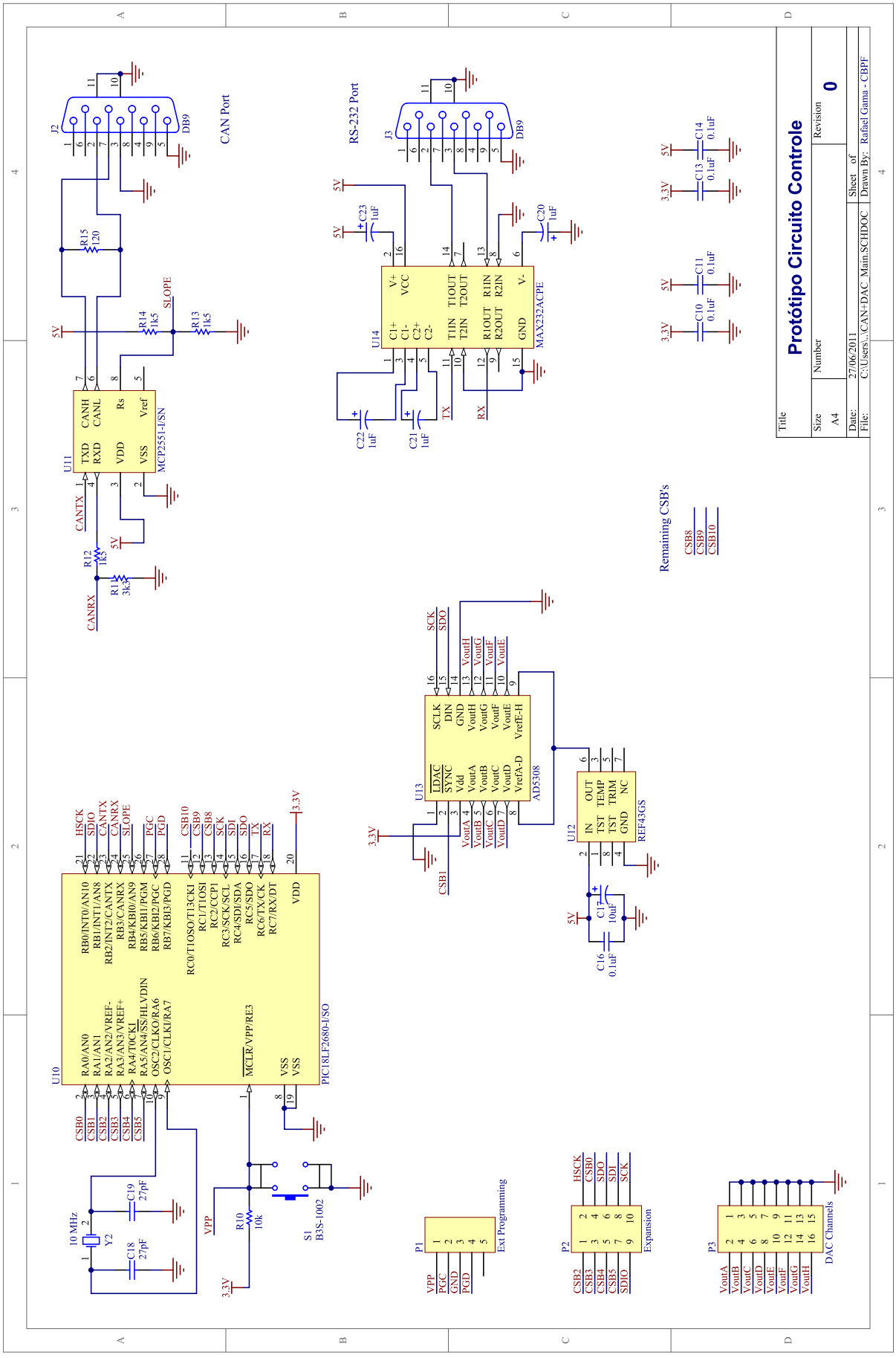
Endereço	Valor	Função
0x300001	0x02	Oscilador HS, monitoramento e chaveamento de *clock* desabilitados
0x300002	0x1F	Temporizador de *power-up* e detector de subtensão desabilitados
0x300003	0x1E	*Watchdog* desabilitado
0x300005	0x80	Pinos da PORTB configurados como e/s digital
0x300006	0x85	*Overflow* ou esvaziamento de pilha provoca *reset*
0x300008	0x0F	Blocos de memória [0-3] desprotegidos
0x300009	0x80	Bloco de *boot* protegido contra escrita
0x30000A	0x0F	Blocos de memória [0-3] desprotegidos contra escrita
0x30000B	0xE0	Registradores de configuração desprotegidos contra escrita
0x30000C	0x0F	Blocos de memória [0-3] desprotegidos contra leitura
0x30000D	0x40	Bloco de *boot* desprotegido contra leitura

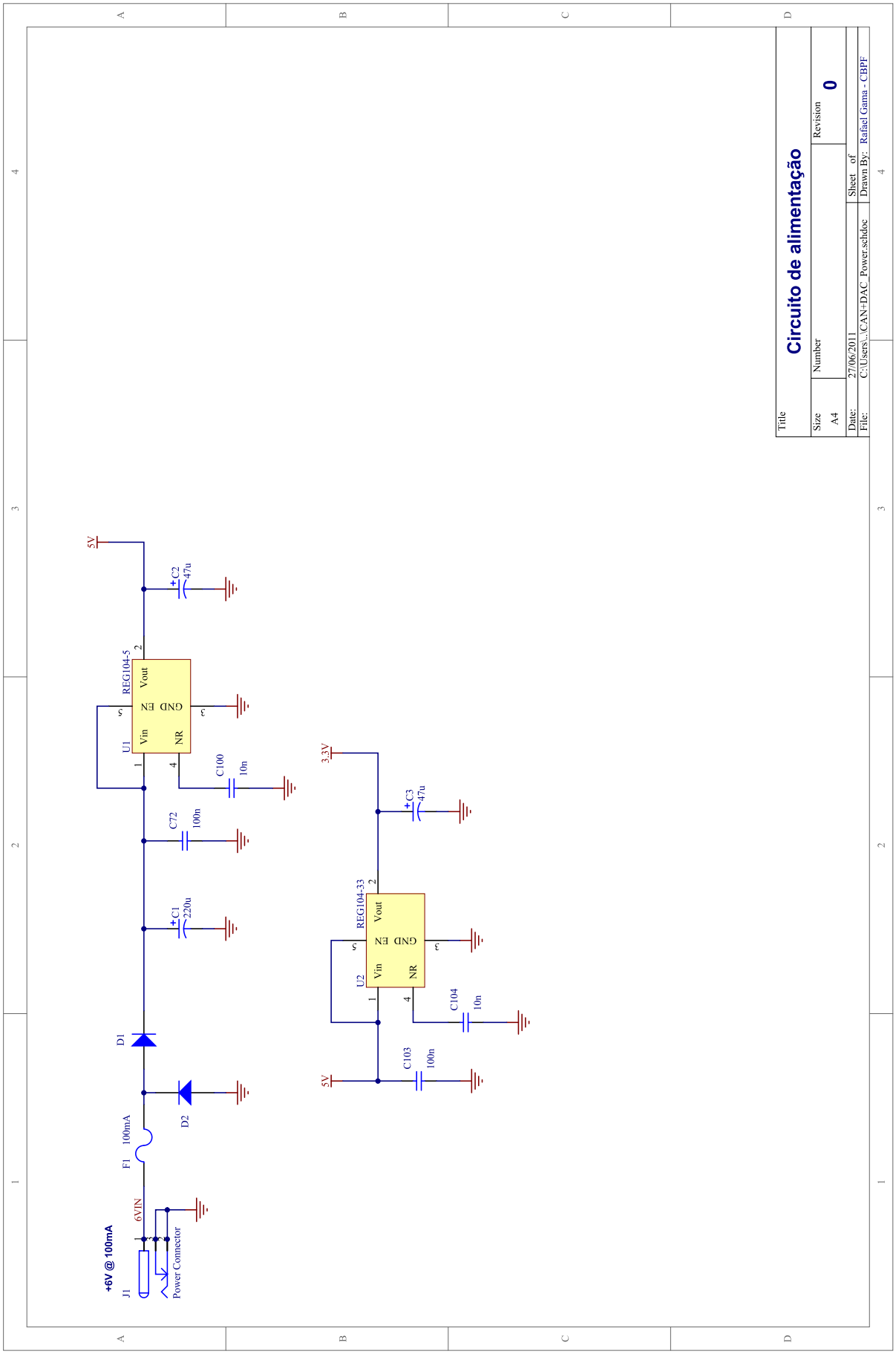
Tabela I: Bits de configuração do microcontrolador

```
// File: 18f2680.lkr
// Sample linker script for the PIC18F2680 processor

LIBPATH .

FILES c018i_800.o
FILES clib.lib
FILES p18f2680.lib

CODEPAGE   NAME=bootloader   START=0x0            END=0x7FF      PROTECTED
CODEPAGE   NAME=vectors      START=0x800          END=0x829      PROTECTED
CODEPAGE   NAME=page         START=0x82A          END=0xFFFF
CODEPAGE   NAME=idlocs       START=0x200000       END=0x200007   PROTECTED
CODEPAGE   NAME=config       START=0x300000       END=0x30000D   PROTECTED
CODEPAGE   NAME=devid        START=0x3FFFFE       END=0x3FFFFF   PROTECTED
CODEPAGE   NAME=eedata       START=0xF00000       END=0xF003FF   PROTECTED

ACCESSBANK NAME=accessram START=0x0              END=0x5F
DATABANK   NAME=gpr0         START=0x60           END=0xFF
DATABANK   NAME=gpr1         START=0x100          END=0x1FF
DATABANK   NAME=gpr2         START=0x200          END=0x2FF
DATABANK   NAME=gpr3         START=0x300          END=0x3FF
DATABANK   NAME=gpr4         START=0x400          END=0x4FF
DATABANK   NAME=gpr5         START=0x500          END=0x5FF
DATABANK   NAME=gpr6         START=0x600          END=0x6FF
DATABANK   NAME=gpr7         START=0x700          END=0x7FF
DATABANK   NAME=gpr8         START=0x800          END=0x8FF
DATABANK   NAME=gpr9         START=0x900          END=0x9FF
DATABANK   NAME=gpr10        START=0xA00          END=0xAFF
DATABANK   NAME=gpr11        START=0xB00          END=0xBFF
DATABANK   NAME=gpr12        START=0xC00          END=0xCFF
DATABANK   NAME=sfr13        START=0xD00          END=0xDFF      PROTECTED
DATABANK   NAME=sfr14        START=0xE00          END=0xEFF      PROTECTED
DATABANK   NAME=sfr15        START=0xF00          END=0xF5F      PROTECTED
ACCESSBANK NAME=accesssfr    START=0xF60          END=0xFFF      PROTECTED

SECTION    NAME=CONFIG     ROM=config

STACK SIZE=0x100 RAM=gpr12
```

Figura 28: Arquivo *Linker Script*

Apêndice B: ESQUEMA ELÉTRICO DO PROTÓTIPO PARA O CIRCUITO DE CONTROLE




Circuito de alimentação
Title
Size
A4
Number
Revision
0
Date: 27/06/2011
File: C:\Users...\CAN+DAC_Power.schdoc
Sheet of
Drawn By: Rafael Gama - CBPF
+6V @ 100mA
J1
Power Connector
6VIN
F1 100mA
D1
D2
5V
C2 47u
U1 REG104-5
Vout
GND EN
Vin NR
C100 10n
C72 100n
C1 220u
3.3V
C3 47u
U2 REG104-33
C104 10n
C103 100n

[1] BARBOSA, A. F. Water cerenkov muon detector near the angra-ii reactor core: the hardware. *Angra Note 007-2009*, 2009.

[2] ANJOS, J. C. Angra dos reis reactor neutrino oscillation experiment. *Brazilian Journal of Physics, vol. 36, p. 1118-1123*, 2006.

[3] GMBH, R. B. *CAN Specification v. 2.0*. [S.l.], 1991.

[4] ASSOCIATION, V. I. T. *American National Standard for VME64*. [S.l.], 1995.

[5] AXELSON, J. *USB Complete*. 3rd. ed. [S.l.]: Lakeview Research LLC, 2006.

[6] DEVICES, A. *Intefacing High Speed ADCs via SPI – AN877*. [S.l.].

[7] LIMITED, A. *Altium Designer*. Http://www.altium.com.

[8] TECHNOLOGY, M. *PIC18F2680 Datasheet – DS39625C*. [S.l.].

[9] TECHNOLOGY, M. *MCP2551 Datasheet – DS21667E*. [S.l.].

[10] TECHNOLOGY, M. *AD5328 Datasheet – Rev. D*. [S.l.].

[11] AB, K. *Kvaser Leaf Userguide*. [S.l.], november 2006.

[12] TERANISH, T. *Tera Term Pro ver 2.3*. Http://hp.vector.co.jp/authors/VA002416/teraterm.html.

[13] AB, K. *Kvaser CANKing*. Http://www.kvaser.com.

[14] TECHNOLOGY, M. *MPLAB C Compiler for PIC18 MCUs – Academic Version*. Http://www.microchip.com.

[15] TECHNOLOGY, M. *MPLAB Integrated Development Environment*. Http://www.microchip.com.

[16] TECHNOLOGY, M. *PIC18C CAN Routines in 'C' – AN738 – DS00738B*. [S.l.].

[17] TECHNOLOGY, M. *Controller Area Network (CAN) Basics – AN713 – DS00713A*. [S.l.].

[18] AB, K. *Kvaser CANlib*. Http://www.kvaser.com/en/developer/canlib.html.

[19] INC., T. *Openchoice Desktop Application*. Http://www.tek.com.

[20] TECHNOLOGY, M. *A Flash Bootloader for PIC16 and PIC18 Devices – AN851 – DS00851*. [S.l.].

[21] DUBUC, M. *Colt PIC18F Bootloader*. Http://mdubuc.freeshell.org/Colt/.