Tese de Mestrado

Contribuição ao desenvolvimento de uma estação de testes para as câmaras de múons do experimento LHCb

Luciano M. de A. Filho

CENTRO BRASILEIRO DE PESQUISAS FÍSICAS
Rio de Janeiro, agosto de 2004

Tese de Mestrado

Contribuição ao desenvolvimento de uma estação de testes para as câmaras de múons do experimento LHCb

Luciano M. de A. Filho

Tese submetida ao Departamento de Campos e Partículas
como requesito para obtenção do grau
de mestre em Física.

Orientador
Ademarlaudo França Barbosa
Co-orientadora
Érica Polycarpo

A minha esposa
Giselle

Agradecimentos

Ao meu orientador, Laudo, por sempre acreditar no meu trabalho e por me proporcionar condições favoráveis para que eu pudesse adquirir toda a carga de conhecimento que absorvi nestes 6 anos de convivência. À minha co-orientadora, Érica, pelo apoio em todos os momentos, pelas preciosas aulas de física de altas energias, e pelos puxões-de-orelha quando eu merecia. Um obrigado especial a Germano e Herman, irmãos com quem aprendi a arte da eletrônica e em quem me espelho para seguir meu rumo nesta estrada intrigante e maravilhosa. É impraticável listar aqui toda a contribuição que eles deram a esta tese! Agradeço também ao Paulo, o mestre em sistemas de detecção, pela ajuda no *setup* com as câmaras de múons. Aos meus amigos, Rogério, Roberto, Rafael e PC, cuja convivência fez do meu trabalho uma verdadeira diversão. A todos os alunos e professores do Mestrado de Instrumentação Científica do CBPF, com quem foi um prazer trabalhar. Finalmente, gostaria de agradecer aos meus pais, que me deram todo o apoio na escolha desta carreira que, infelizmente, ainda é muito desvalorizada no nosso país, mas sem a qual eu não seria feliz, jamais!

Resumo

O presente trabalho é uma contribuição ao desenvolvimento de uma estação para testar as câmaras de fios (MWPCs) do *Sistema de Múons* do experimento LHCb. São necessárias cerca de 1400 destas câmaras, operando dentro de uma mesma faixa de desempenho, para compor o sistema. Por isso, um rigoroso processo de teste e caracterização deve ser aplicado a estas câmaras antes que elas possam ser instaladas no detector. A estação proposta utiliza o método de linhas-de-retardo para estimar a distribuição espacial da eficiência, e medidas relativas de ganho para teste de uniformidade da câmara. Utiliza-se, para processar os sinais das linhas-de-retardo, um conversor tempo-digital de 8 canais com resolução de 120 ps. Para a medida de ganho foi projetado um analisador multicanal com um conversor analógico-digital *pipeline* de 40 MHz e resolução de 2 mV.

Abstract

The present work is a contribution to the development of a test station for the wire chambers (MWPCs) used in the muon system of the LHCb experiment. 1400 chambers must operate under the same performance in order to compose the system. A rigorous test and characterization process should be applied to these chambers before their installation in the detector. The station proposed here makes use of the delay-line method to estimate the spatial distribution of efficiency, and of the measurement of gain to test the uniformity of the chamber. The test station includes an 8-channel time to digital converter with 120 ps resolution. For the gain measurement a multichannel-analyzer has been developed with a 40 MHz pipeline analog to digital converter with 2 mV resolution.

Conteúdo

Lista de Figuras

Introdução

O LHCb é um dos experimentos aprovados para o colisor LHC, previsto para operar no CERN a partir de 2007. Seu principal objetivo é o estudo da violação de carga-paridade (CP), para uma melhor compreensão, entre outros temas, da discrepância entre quantidade de matéria e anti-matéria encontradas no universo. A violação de CP é observada em decaimento dos mésons-B, presentes nas colisões próton-próton do LHC. Múons são produzidos no estágio final deste decaimento e detectados pelo *Sistema de Múons* do LHCb. Este sistema, por sua vez, é formado por cinco estações, preenchidas com câmaras MWPCs, com leitura de anodo e catodo, que atuam na tomada de decisão do *trigger* do LHCb e na reconstrução da trajetória destas partículas. São necessárias cerca de 1400 destas câmaras para cobrir uma área total de 435 m^2 do *Sistema de Múons*.

Devido ao número elevado de câmaras a serem produzidas, faz-se necessário um procedimento padrão de teste e caracterização das mesmas, antes que elas possam ser instaladas no experimento. Foi desenvolvida, no CBPF, uma *Estação de Testes* com esta finalidade. Tal estação utiliza a técnica de linhas-de-retardo para identificação dos *pads* de catodo relacionados com a posição de uma partícula detectada. Desta forma é possível medir variações da eficiência ao longo dos planos da câmara, contando-se o número de eventos por *pad*, enquanto a câmara é iluminada por uma fonte de radiação homogênea. A estação também utiliza a informação de carga gerada no anodo para uma análise da uniformidade de ganho. O procedimento de teste é realizado, aplicando-se alta tensão à câmara e adquirindo-se dados com raios cósmicos ou iluminando-a com uma fonte radioativa.

O presente trabalho é uma contribuição ao desenvolvimento desta estação no que

roncerne ao processamento, armazenamento e análise dos dados. O *Módulo de Processamento de Dados* (MPD) da estação é composto de uma parte em *hardware* e outra em *software*. O *hardware* utiliza um circuito integrado conversor tempo-digital (TDC) de oito canais, com resolução de 120 ps por canal, para medidas de intervalo de tempo entre sinais provenientes das linhas-de-retardo. O MPD conta também com um conversor amplitude-digital (ADC) do tipo *pipeline*, para amostragem do sinal de anodo a uma taxa de 40 MHz, com uma resolução de 2 mV. Os valores convertidos, tanto em tempo quanto em amplitude, são histogramados e armazenados em memórias locais no MPD. O controle e processamento digital dos dados é feito por dispositivos lógico-programáveis do tipo FPGAs. Um interfaceamento, via porta-paralela, faz a comunicação entre o *hardware* e um PC (*Personal Computer*).

A parte em *software* tem por objetivos: controlar o processo de aquisição, interpretar os dados histogramados e gerar arquivos com os resultados da caracterização. Tais procedimentos são realizados por um único programa denominado *Chambers 1.0*. Este programa, desenvolvido em *Borland Delphi 7.0*, utiliza a característica modular de uma linguagem orientada a objetos para desenvolvimento de classes específicas que controlam desde o acesso ao *hardware* até a geração de gráficos para interface com o usuário. O *Chambers 1.0* gera relatórios com os resultados finais de caracterização em formato *Microsoft Word* (.doc).

O primeiro capítulo é uma introdução ao LHC, com maior ênfase ao experimento LHCb, onde é destacada a importância das câmaras de múons. Este capítulo termina com uma descrição técnica destas câmaras e uma introdução ao seu princípio de funcionamento. O Capítulo 2 trata da *Estação de Testes* propriamente dita. Primeiramente são apresentados os parâmetros da câmara que devem ser medidos, seguida pela descrição das técnicas utilizadas para se medir estes parâmetros. Por último, este capítulo apresenta a estação completa e descreve a função de cada um de seus componentes. Os Capítulos 3 e 4 descrevem, em detalhes, o funcionamento da parte em *hardware* e em *software* do MPD, respectivamente. O último capítulo apresenta medidas de desempenho feitas com o MPD e resultados obtidos com a *Estação de Testes* em uma câmara protótipo.

Capítulo 1

O Projeto LHC

O colisionador LHC (*Large Hadron Collider*) [1] está em fase de construção no CERN, em Genebra, na Suíça. Ele consistirá de um tubo em um túnel, contendo 8.000 magnetos supercondutores ao longo de uma circunferência de 26.7 km, no qual dois feixes de prótons, circulando em direções opostas, colidirão a uma taxa de 40 milhões por segundo.

Para produzir, armazenar e acelerar os prótons, o túnel do LEP [2] e toda a infraestrutura ao seu redor serão usados. O sistema de injeção é mostrado na Figura 1-1. Os prótons são acelerados até uma energia de 50 MeV no acelerador linear (*proton LINAC - LINear Accelerator*) e, em seguida, injetados no PSB (*Proton Synchrotron Booster*), de onde sairão com uma energia de 1.4 GeV para o PS (*Proton Synchrotron*). No PS e no SPS (*Super Proton Synchrotron*), as energias finais dos prótons serão de 26 GeV e 450 GeV, respectivamente. Finalmente, no LHC, ocorrerá a aceleração final até a energia de 7 TeV.

Quatro experimentos foram aprovados para o LHC:

1. **CMS** (*Compact Múon Solenoid* [3])

   Experimento de propósito geral para estudo do bóson de Higgs, de partículas supersimétricas e de física de íons pesados.

2. **ATLAS** (*A Toroidal LHC ApparatuS* [4])

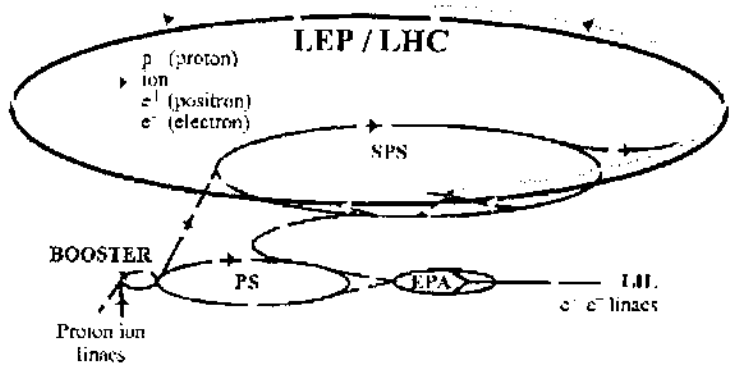

Figura 1-1: LHC - sistema de injeção [1].

Também é um experimento de propósito geral que armazenará colisões próton-próton no LHC. Foi otimizado para ter o máximo alcance possível da física proposta pelo LHC.

3. **ALICE** (*A Large Ion Collider Experiment* [5])

   O ALICE é o único experimento do LHC inteiramente dedicado à física de colisões nucleares. Seu principal objetivo é estabelecer e estudar a formação do plasma de quarks e glúons.

4. **LHCb** (*Large Hadron Collider beauty experiment for precision measurements of CP-violation and rare decays* [6])

   É um experimento dedicado ao estudo de violação de CP e outros fenômenos raros, oriundos do decaimento do méson-B.

O presente trabalho está relacionado com o experimento LHCb, que é descrito com mais detalhes a seguir.

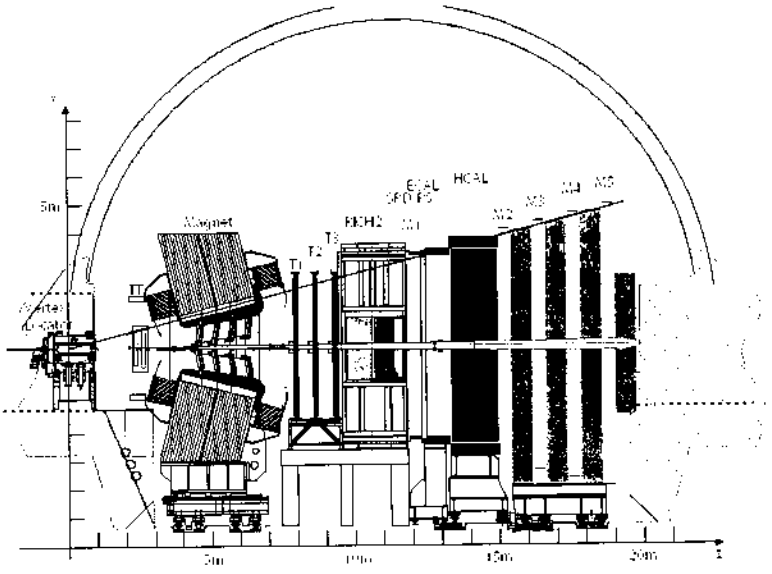

Figura 1-2: Vista lateral do detector do LHCb [7].

1.1 O experimento LHCb

O detector LHCb, ilustrado nas Figuras 1-2 e 1-3, é um espectrômetro de geometria similar aos experimentos de alvo fixo. Sua geometria foi escolhida em função da cinemática de produção dos pares $b\bar{b}$ em colisões próton-próton. Proposto em 1998, tem como objetivo realizar medidas de violação de CP e de fenômenos raros no decaimento do méson-B. O comprimento total do detector de, cerca de 20 metros, é limitado pelo tamanho da caverna no ponto IP8 onde está instalado. Durante o funcionamento do LEP, esta caverna era ocupada pelo experimento DELPHI [8].

1.1.1 O detector de vértice

O VELO (*VErtex LOcator*) deve fornecer informações precisas sobre as coordenadas dos traços próximos ao ponto de interação. Ele deve também fornecer informação para o segundo nível de *trigger*. A tecnologia utilizada para o sistema é a de detectores de microtiras de silício com 300 μm de espessura. Detalhes podem ser encontrados no

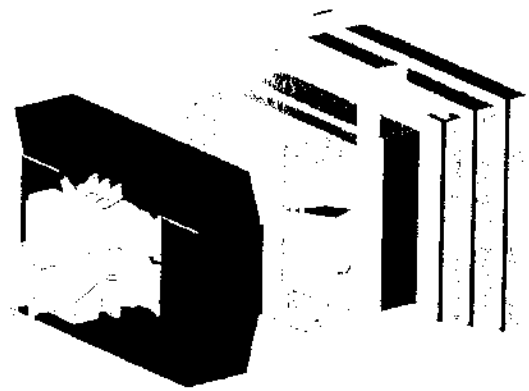

Figura 1-3: Visão em perspectiva do detector do LHCb [7].

Technical Design Report do detector de vértices [9].

1.1.2 O sistema de veto de empilhamento

Este detector, localizado junto ao VELO, deve rejeitar eventos com mais de um vértice primário e passar esta informação ao primeiro nível de *trigger*. Sua leitura deve, portanto, ser realizada dentro de 25 ns, intervalo de tempo entre as colisões das nuvens de prótons do LHC.

1.1.3 Ímã e sistema de determinação de trajetórias

O sistema de *tracking* [10] deve permitir a reconstrução das trajetórias de partículas carregadas com eficiência de 98.5% para os detectores internos de silício e 94% para os detectores externos de *straw tubes* [10], para partículas de momento acima de 10 GeV. O raio de separação entre os dois subsistemas é de 50 cm. Este sistema deve determinar, ainda, o momento da partícula com alta precisão. Para que esta medida tenha uma precisão relativa melhor que cerca de 0.5%, para partículas de momento até 200 GeV, o ímã deve fornecer um campo magnético integrado de 4 Tm. Estudos mostraram que

para uma melhor relação custo-benefício um ímã não-supercondutor deve ser usado [11].

1.1.4 Detectores RICH

Dois detectores RICH (*Ring Imaging CHerenkov*) estão sendo desenvolvidos para identificar partículas [12], tarefa fundamental para a supressão de *background*. O primeiro detector RICH identificará partículas de 1 a 70 GeV, e o segundo permitirá a identificação de partículas de até 100 GeV. A associação de anéis nos dois detectores RICH combinada com as trajetórias reconstruídas no sistema de *tracking* determinarão probabilidades para as hipóteses de a partícula ser elétron, múon, píon, káon ou próton.

1.1.5 Calorímetros

A função dos calorímetros é contribuir para a identificação de hádrons, elétrons e fótons, e medir suas energias e posições. A energia e posição das células ativadas são a base para o primeiro nível de *trigger*. Sua composição é dada por um *Scintillator Pad Detector* (SPD), seguido de um *preshower* (PS), de um calorímetro eletromagnético (ECAL) e de um calorímetro hadrônico (HCAL). O SPD e o PS foram introduzidos para reduzir a contaminação no *trigger* por píons neutros e carregados, respectivamente. O ECAL tem espessura de 25 comprimentos de radiação e o HCAL 5.6 comprimentos de interação. O comprimento total de interação é de 6.8 [13].

1.1.6 Sistema de *Trigger*

O sistema de *trigger* do LHCb é composto de quatro níveis. O nível 0 utiliza informação dos calorímetros e do sistema de múons para selecionar elétrons, múons e hádrons de alto momento transverso. Ele também rejeita eventos com mais de uma interação próton-próton a partir da informação do sistema de veto de empilhamento. Este nível opera a uma frequência de 40 MHz, reduzindo esta taxa a um valor médio de 1 MHz [14] com um tempo de latência máximo de 4 μs. O nível 1 dispara quando reconhece vértices separados. Estes traços são, então, combinados, se estiverem próximos. A sua latência máxima é de

1024 μs e a taxa de informação é reduzida para 40 kHz [15]. Os níveis 2 e 3 reduzem esta taxa para 5 kHz e 200 Hz, respectivamente. O nível 2 utiliza a informação de momento, fornecida pelo sistema de *tracking*, para eliminar falsos vértices secundários. O nível 3 utiliza informação de todos os subdetectores do LHCb para selecionar decaimentos de hádrons-b com base nos códigos de reconstrução e análise.

As câmaras de múons, que são de principal interesse para o presente trabalho, atuam no nível 0 de *trigger*, e formam o chamado *Sistema de Múons* [7], que será descrito com mais detalhes no próximo item. Boas descrições das técnicas utilizadas pelos subdetectores do LHCb são encontradas nas referências [16], [17], [18] e [19].

1.2 O Sistema de Múons

O *Sistema de Múons* deve selecionar múons de alto momento transverso para o *trigger* de nível 0, além de fazer a reconstrução *offline* de múons. Ele é composto de cinco estações (Figura 1-2). A estação M1 fica a 12.1 metros da região de colisão do feixe e, junto com a estação M2, permite a medida do momento transverso da partícula para tomada de decisão do nível 0 de *trigger*. As estações M2, M3, M4 e M5 ficam a uma distância média do ponto de interação de 15.2, 16.4, 17.6 e 18.8 metros, respectivamente. Elas são intercaladas por paredes de ferro de 80 cm de espessura que funcionam como filtro. Uma vista lateral do sistema, cuja área total é de 435 m^2, é mostrada na Figura 1-4. Cada caixa representa uma câmara de múon. Os elementos da câmara, cuja leitura é feita por canais de *front-end* independentes, são agrupados para formar canais lógicos de leitura. O esquema lógico descreve a granularidade de leitura nas direções **x** e **y** em cada região das estações do *Sistema de Múons*, assim como ela é vista pelo *trigger* e pelo algoritmo de reconstrução *offline*.

O processo de reconstrução do *trigger* de múons ocorre da seguinte forma (Figura 1-5): para cada bloco lógico ativado em M3, procura-se por eventos em M2, M4 e M5, em uma região de interesse em torno de uma linha, projetada ao ponto de interação. Se houver eventos em todas as estações, é feita uma extrapolação para M1 a partir dos

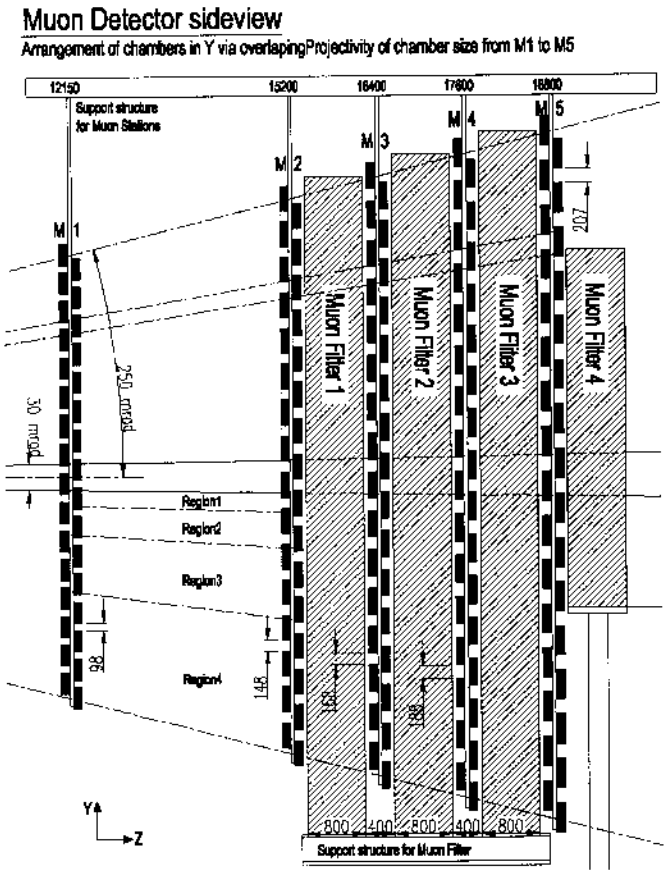

Figura 1-4: Vista lateral do *Sistema de Múons* do LHCb [7].

blocos ativados de M2 e M3, e escolhido o bloco ativado em M1 no ponto mais próximo da projeção extrapolada. A direção dos traços indicada pelos *hits* em M1 e M2 é usada na medida do momento transverso, supondo-se que a partícula sofre um impulso em um ponto fixo localizado no centro de curvatura do dipolo. A implementação do nível 0 de *trigger* de múons é explicada em detalhes nas referências [20] e [21].

O papel das câmaras de múons é, portanto, fundamental no experimento LHCb. Algumas destas câmaras serão testadas pela estação proposta neste trabalho antes de serem instaladas no experimento. No próximo item é apresentada uma descrição técnica destas câmaras, bem como de seu princípio de funcionamento, antes de abordarmos a

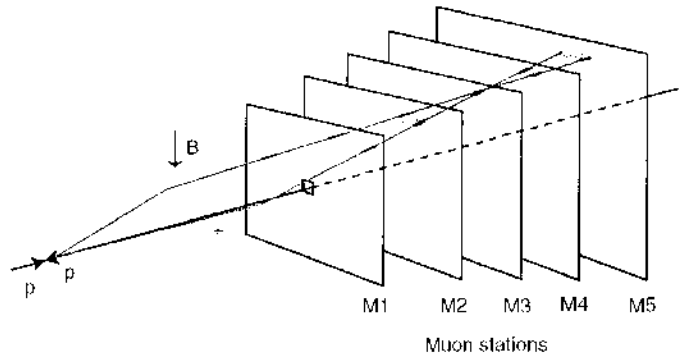

Figura 1-5: Processo de reconstrução do *trigger* de múon do LHCb. O algoritmo utiliza informação das cinco estações para estimar a trajetória da partícula [7].

Estação de Testes propriamente dita, no próximo capítulo.

1.3 A Câmara de Múons

Optou-se, como tecnologia a ser adotada em todo o *Sistema de Múons*, por câmaras do tipo MWPCs (*Multi Wire Proportional Counters*), com exceção da região central da estação M1, devido ao fluxo muito alto de partículas nesta região. Como ela corresponde a menos de 1% da área total do detector, decidiu-se pelo adiamento da definição da tecnologia a ser utilizada nesta parte do sistema.

1.3.1 Princípio de operação das câmaras MWPCs

O processo de detecção de múons, como o de qualquer outra partícula, dá-se pela interação entre esta e o meio detector, fazendo com que a partícula perca energia para o meio. Tal perda pode resultar no desprendimento de elétrons de átomos do meio, criando pares e^--íons$^+$. Estes pares podem ser coletados, por exemplo, aplicando-se uma diferença-de-potencial entre dois eletrodos. A detecção da partícula é realizada de forma indireta através da medida de um sinal elétrico nos eletrodos. Neste processo é mais

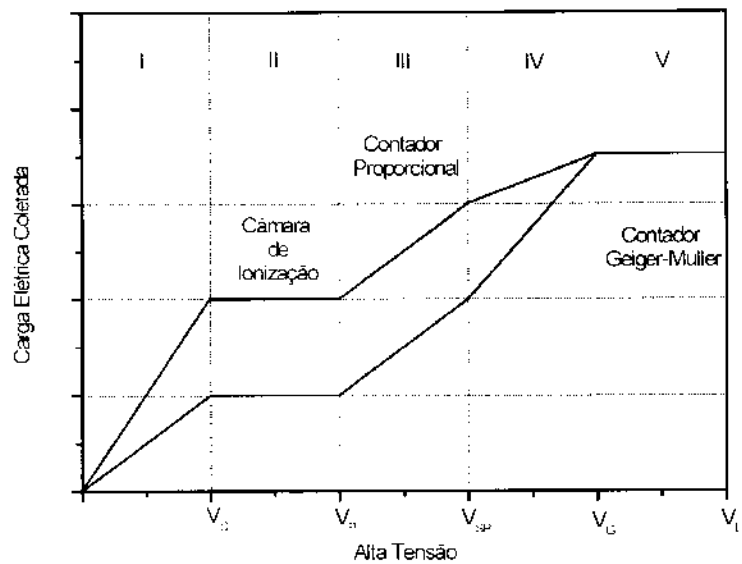

Figura 1-6: Regiões de operação possíveis para um detector a gás [22].

comum a utilização de gás como meio detector, justamente porque é mais fácil coletar carga elétrica em um gás do que em um sólido ou em um líquido.

A relação entre a carga elétrica coletada por evento e a diferença-de-potencial aplicada entre os eletrodos que contêm o gás, apresentada de forma qualitativa na Figura 1-6, mostra que podemos definir cinco regiões distintas de operação [22].

1. Região I - Devido ao baixo potencial nesta região, muitos pares e^--íons$^+$ se recombinam no meio antes de serem coletados, e, a probabilidade de pares alcançarem os eletrodos aumenta com o potencial.

2. Região II - O detector atinge uma saturação, onde todos os pares são coletados. Detectores que operam nesta região são conhecidos como *Câmaras de Ionização.*

3. Região III - Nesta região, ocorre um fenômeno conhecido como *multiplicação gasosa*, onde pares gerados por partículas incidentes adquirem energia suficiente para gerar outros pares e assim por diante. A carga gerada pela multiplicação gasosa é proporcional à perda de energia da partícula ionizante. Por esta razão, tais detectores

são ditos proporcionais.

4. Região IV - O grande número de pares produzido pela multiplicação gasosa resulta na formação de uma distribuição espacial de carga entre os eletrodos, diminuindo o campo elétrico efetivo. Isto conduz a uma perda gradativa da proporcionalidade.

5. Região V - A carga coletada se torna independente da carga de ionização primária, e a multiplicação gasosa se propaga ao longo do eletrodo anódico. Detectores que operam nesta região são chamados de *Contadores Geiger-Müller*.

1.3.2 Câmaras de múons do LHCb

Câmaras MWPCs, como as do *Sistema de Múons*, utilizam anodos em forma de fios com espaçamento de 2 mm e operam na região proporcional. A composição do gás é Ar (40%), CO_2 (45%), CF_4 (15%). A Figura 1-7 mostra a configuração de uma câmara de múons do LHCb, com seus quatro planos de detecção. Nas Figuras 1-8 (a) e (b) são mostrados alguns detalhes dos planos catódico e anódico, respectivamente. Os planos catódicos são divididos em 64 *pads*, sendo cada *pad* ligado eletricamente a um conector situado em uma das laterais da câmara. Cada conector do plano catódico permite a leitura de 8 *pads*. No plano anódico, os fios, eletricamente conectados em grupos de oito, são ligados a um barramento de alta voltagem, via resistores limitadores de corrente. A leitura do sinal anódico pode ser feita pelo lado oposto aos resistores. O *gap* entre catodo e anodo é de 2.5 mm. A Figura 1-9 mostra um painel de uma destas câmaras. O conjunto de informações técnicas sobre as câmaras de múons é apresentado em [23].

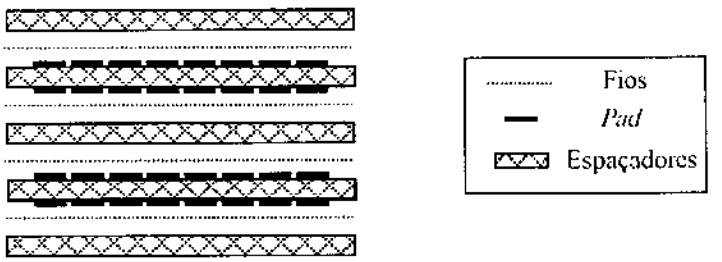

Figura 1-7: Esquema de configuração das câmaras de múons do LHCb, com quatro planos de detecção.

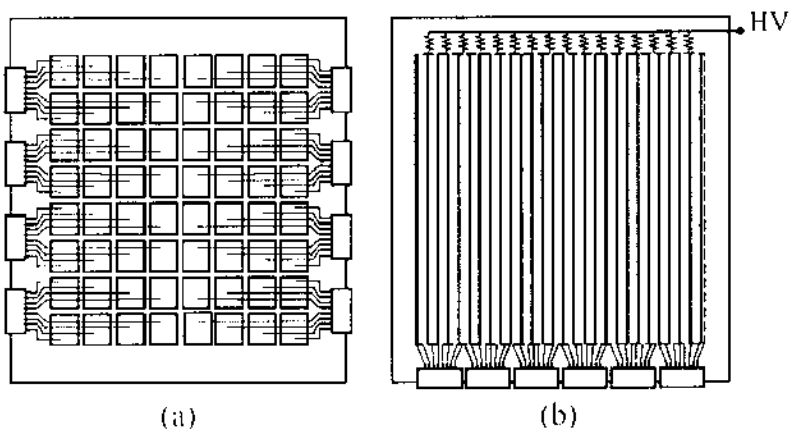

Figura 1-8: Detalhes dos planos catódico (a) e anódico (b) das câmaras de múons do LHCb. Os *pads* de catodo e os fios de anodo são acessados por conectores nas laterais e na parte inferior da câmara, respectivamente.

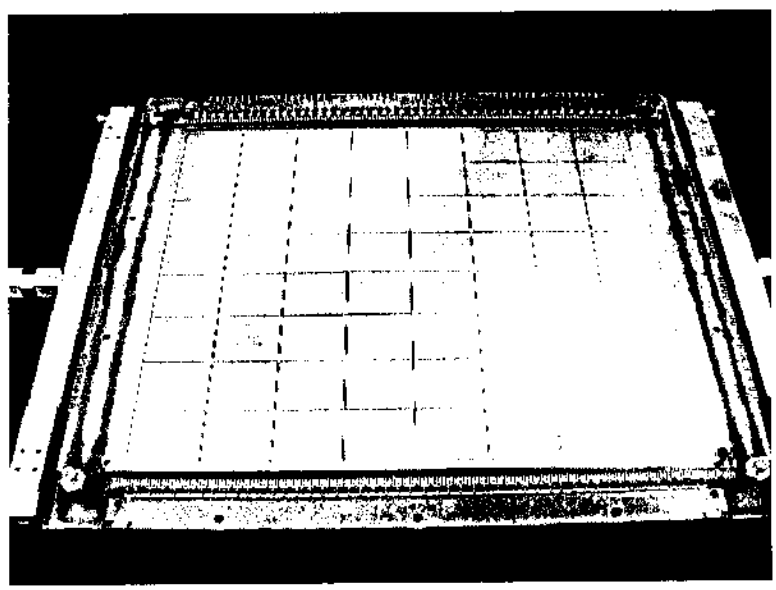

Figura 1-9: Foto de um dos painéis da câmara de múons do LHCb.

Capítulo 2

A Estação de Testes

As estações M1 a M5 do *Sistema de Múons* serão preenchidas com cerca de 1400 câmaras MWPCs (§ 1.2). Devido ao número elevado, faz-se necessário um processo de qualificação destas câmaras, antes que elas possam ser usadas no LHCb. Este capítulo apresenta a *Estação de Testes,* projetada com esta finalidade.

Os parâmetros da câmara a serem medidos pela *Estação de Testes* estão relacionados com sua eficiência. Entende-se por eficiência a razão entre a quantidade de partículas detectadas e a quantidade de partículas incidentes na câmara. Este parâmetro deve ser superior a 99% em uma janela de 20 ns. Conforme apresentado no capítulo anterior, as câmaras são construídas com 4 planos anódicos. Com isto, pode-se "relaxar" a eficiência para cerca de 95% em cada plano. Contudo, o sistema será mais redundante e imune a flutuações de desempenho do detector, se cada plano tiver, também, eficiência de 99%. Este fato já foi comprovado com medidas em *test-beams* [23]. A estação aqui proposta se destina a medir parâmetros que dão um maior grau de detalhamento na caracterização das câmaras. São eles: a eficiência relativa entre *pads*, que é uma medida relativa da distribuição espacial da eficiência, e a uniformidade de ganho, que relaciona a quantidade de carga gerada com a energia depositada pela partícula incidente em cada *pad*. Este último parâmetro está diretamente relacionado com a eficiência da câmara, como será mostrado a seguir.

2.1 Parâmetros de caracterização

2.1.1 Eficiência relativa entre *pads*

Sabe-se, a partir de medidas feitas em *test-beams* com câmaras protótipo, que a eficiência da câmara é próxima de 99%. Contudo, deseja-se quantificar variações deste parâmetro ao longo do plano da câmara, ou seja, medir sua distribuição espacial. Isto pode ser feito, computando a eficiência relativa entre *pads*. O procedimento consiste em contar eventos por *pad* ao se iluminar a câmara uniformemente com uma fonte radioativa ou com raios cósmicos. Uma comparação entre a quantidade total de eventos acumulados em cada *pad* fornece um resultado quantitativo da eficiência relativa entre *pads*.

2.1.2 Ganho e uniformidade

O ganho da câmara é dado pela relação entre a quantidade de carga detectada e a energia depositada no detector. Este parâmetro é importante devido ao modo como a eletrônica de *front-end* determina a ocorrência de um evento válido: pela comparação da tensão gerada nos eletrodos com um valor de *threshold*, abaixo do qual há apenas ruído eletrônico. Portanto, um evento considerado válido em uma região do detector, pode não o ser em uma outra região, do mesmo detector, que tenha ganho ligeiramente inferior, ou seja, a eficiência varia com o ganho.

Se a medida for feita com a câmara uniformemente iluminada, tal parâmetro é denominado *ganho global*. No entanto, utilizando uma fonte colimada, de modo que apenas um *pad* seja iluminado por vez, pode-se fazer o mapeamento do ganho para cada *pad* de catodo. Desta forma, pode-se medir a *uniformidade de ganho* da câmara. A medida da uniformidade de ganho requer a utilização de uma fonte radioativa, para varrer os 64 *pads* de catodo.

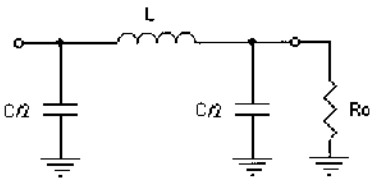

Figura 2-1: Célula de retardo, tipo π, com uma terminação R_0.

2.2 Tecnologia e procedimentos

Para caracterização dos parâmetros acima descritos, a estação faz uso da seguinte tecnologia e procedimentos:

2.2.1 Identificação de *pads* pela técnica de linha-de-retardo

Linha-de-retardo é um dispositivo que introduz retardo à propagação de sinais. Podem-se realizar diversas configurações, utilizando-se capacitores e indutores de modo a formar células de retardo que, quando conectadas em cascata, formam linhas-de-retardo ditas "discretas". Uma célula de retardo simples é o filtro LC da Figura 2-1. Para que não haja reflexão de componentes espectrais do sinal, utiliza-se, como carga, um resistor R_0 de valor tal que

$$R_0 = \sqrt{\frac{L}{C}} \frac{1}{\sqrt{1-\nu^2}}\bigg|_{\nu \to 0} = \sqrt{\frac{L}{C}} \tag{2.1}$$

onde

$$\nu = \frac{w}{w_0} \tag{2.2}$$

$$w_0 = \frac{2}{\sqrt{LC}} \tag{2.3}$$

e w é a frequência de componentes espectrais. O termo central da Equação 2.1 é a terminação ideal para que haja um perfeito casamento de impedância no circuito da Figura 2-1 [22]. Para $\nu \to 0$, vê-se que R_0 se reduz a um valor constante. Utilizando este valor para R_0 obtém-se a seguinte relação entre as tensões de entrada e de saída, em

função da frequência (função de transferência):

$$T = \frac{1}{1 - 2\nu^2 + 2i\nu} = A(\nu)e^{i\varphi(\nu)} \tag{2.4}$$

onde

$$A(\nu) = \frac{1}{\sqrt{1 + 4\nu^2}} \tag{2.5}$$

$$\varphi(\nu) = -tg^{-1}\left(\frac{2\nu}{1 - 2\nu^2}\right) \tag{2.6}$$

que para $\nu \rightarrow 0$ tornam-se

$$A = 1 \tag{2.7}$$

$$\varphi = -2\nu = -w\sqrt{LC} \tag{2.8}$$

ou seja, o circuito tem ganho 1 e fase linear (Figura 2-2). As Equações 2.7 e 2.8 sugerem que um harmônico de frequência w da forma

$$A_{entrada} = A\cos(wt + \phi) \tag{2.9}$$

será defasado ao percorrer a célula de retardo, de modo que o sinal na saída da célula é

$$A_{saída} = A\cos\left(wt + \phi - w\sqrt{LC}\right) \tag{2.10}$$

Pode-se escrever a Equação 2.10 como

$$A_{saída} = A\cos\left[w\left(t - \sqrt{LC}\right) + \phi\right] \tag{2.11}$$

de onde se torna evidente que o retardo causado pela célula é independente da frequência e vale

$$\tau = \sqrt{LC} \tag{2.12}$$

para $\nu \rightarrow 0$. Portanto, os valores de L e C da célula devem ser escolhidos de modo a satisfazer a banda do sinal aplicado (Equação 2.3) e o retardo desejado (Equação

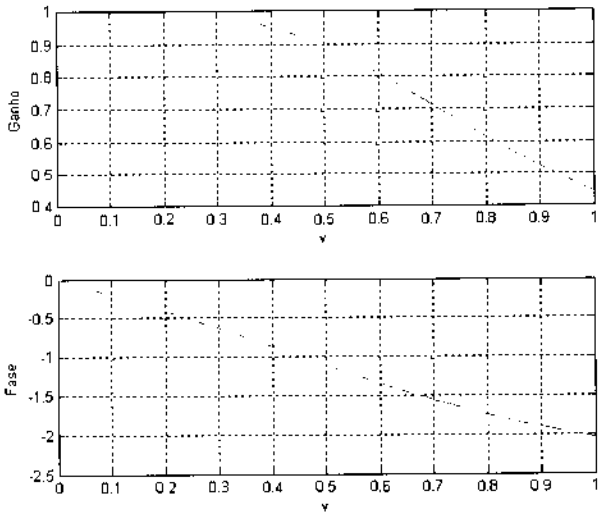

Figura 2-2: Gráfico da função de transferência de uma célula de retardo tipo π, para frequência normalizada ν. Observa-se que o ganho é constante e a fase é linear para $\nu \rightarrow 0$.

2.12) simultaneamente. O valor do resistor de terminação é dado pela Equação 2.1. A associação de n células de retardo leva a um circuito, conhecido como *linha-de-retardo*, que tem função de transferência T^n e retardo total de $n\tau$.

Cada *pad* de catodo da câmara é conectado a uma célula de uma linha-de-retardo (§ 2.3.1). Desta forma os *pads* são identificados, medindo-se o tempo que um sinal, nele gerado, leva para alcançar uma das extremidades da linha. Um histograma feito com esta informação pode ser utilizado para medir a eficiência relativa entre *pads*, e, consequentemente, estimar a distribuição espacial da eficiência da câmara, caso ela seja iluminada uniformemente. Esta técnica também identifica defeitos mais grosseiros como, por exemplo, a não ligação de um *pad* ao seu respectivo conector, ou um curto-circuito entre *pads* vizinhos.

2.2.2 Leitura do sinal de anodo por desacoplamento capacitivo

A forma do sinal de anodo é essencial para obtermos o parâmetro de ganho da câmara. Pode-se demonstrar que o sinal elétrico formado por uma carga q, coletada entre dois eletrodos, formados por um fio e um cilindro, centrados no mesmo eixo, é da forma [22]:

$$u^{+}(t) = -\frac{q}{2\pi\varepsilon\varepsilon_0} \ln\left(\sqrt{1 + \frac{t}{t_0}}\right) \tag{2.13}$$

em que

$$t_0 = \frac{p\pi\varepsilon\varepsilon_0 r_0^2}{\mu^{+} C_e V_0} \tag{2.14}$$

onde p é a pressão do gás, r_0 a distância radial onde o evento ocorreu, μ^{+} a constante de mobilidade dos íons positivos, V_0 a tensão aplicada entre os eletrodos e C_e a capacitância entre os eletrodos. Como o diâmetro do fio anodo é muito pequeno (30 μm), comparado a distância anodo-catodo, a Equação 2.13 é uma boa aproximação prática para o sinal gerado nos eletrodos da câmara, mesmo quando o catodo não tem forma cilíndrica. Tal sinal deve ser desacoplado da alta tensão antes de ser analisado. A Figura 2-3 mostra um modelo AC para o sinal no anodo desacoplado pelo capacitor C_d e coletado na carga R. R_h representa a resistência limitadora de corrente do fio anódico (ver Figura 1-8 b) e C é a capacitância entre os eletrodos. Tipicamente, $C_d \gg C$ e $R_h \gg R$. De modo que este circuito pode ser reduzido à representação da Figura 2-4. Pode-se demonstrar [22] que

$$v_{saída} = u^{+}(t) e^{\frac{-t}{RC}} \tag{2.15}$$

A Equação 2.15 dá a forma do pulso coletado no anodo. A Figura 2-5 mostra alguns destes pulsos para diferentes valores de constante de tempo RC.

Na câmara de múons do LHCb, este sinal é obtido curto-circuitando-se todos os fios de um plano de anodo através dos conectores disponíveis na câmara. A Figura 2-6 mostra um esquema deste procedimento.

A informação sobre a quantidade de carga, induzida nos eletrodos da câmara, está

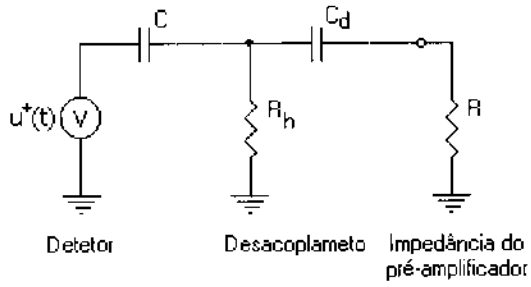

Figura 2-3: Modelo AC para o sinal desacoplado no anodo de uma câmara de fios.

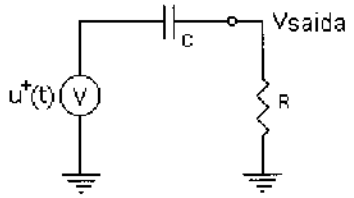

Figura 2-4: Circuito equivalente ao da Figura 2-3 quando $C_d \gg C$ e $R_h \gg R$.

relacionada com a forma do pulso $v_{saída}$ capturado no anodo pela equação

$$q = \frac{1}{R} \int v_{saída} dt \tag{2.16}$$

Portanto, o parâmetro de ganho pode ser estimado pela razão entre a integral do pulso de tensão e a impedância de carga R. Este procedimento é realizado por um analisador multicanal projetado para este fim (§ 2.2.4).

2.2.3 Medidas de tempo utilizando TDCs

TDCs (*Time to Digital Converter*), como o próprio nome sugere, são dispositivos que convertem intervalos de tempo em uma palavra digital. As técnicas mais conhecidas para a implementação de TDCs são:

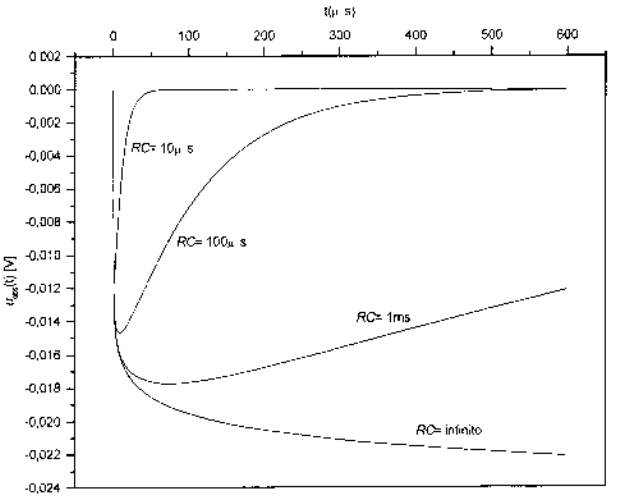

Figura 2-5: Pulsos de anodo coletados nos terminais de uma câmara de fios, para diferentes valores de R e C do circuito equivalente de desacoplamento.

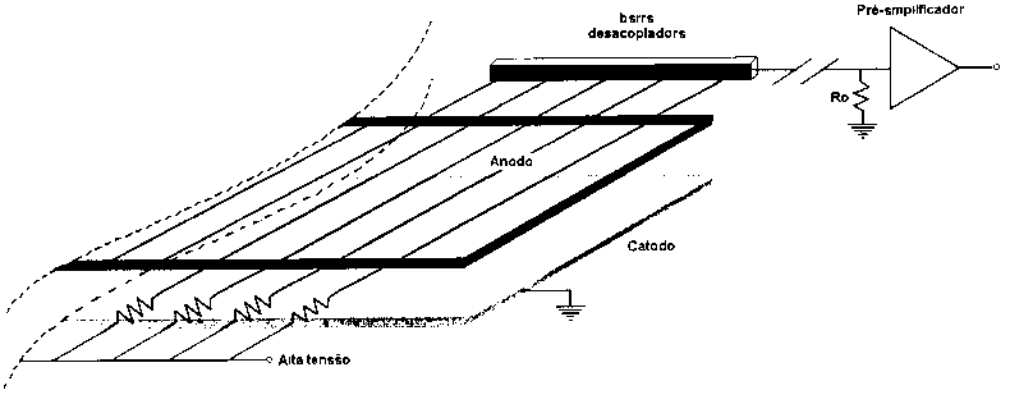

Figura 2-6: Esquema de desacoplamento capacitivo do sinal de anodo.

2.2.3.1 Técnica da integração de corrente

Nesta técnica, um capacitor é carregado linearmente com uma corrente constante durante o intervalo de tempo entre os pulsos de controle (*start* e *stop*) [24]. A carga acumulada no capacitor e, portanto, a tensão em seus terminais, é proporcional a este intervalo de tempo. Posteriormente, um conversor analógico-digital (Analog to Digital Converter - ADC) é utilizado para converter esta tensão em uma palavra digital. A resolução, neste tipo de conversor, pode atingir a faixa de ps. A estabilidade da fonte de corrente, a linearidade do capacitor e a resolução do ADC determinam a resolução temporal que pode ser alrançada com esta técnica.

2.2.3.2 Contadores

Nesta técnica, um contador de alta freqüência é habilitado durante o intervalo de tempo entre os pulsos de *start* e *stop* [25]. A palavra digital, correspondente à medida de tempo, é obtida diretamente do barramento de saída do contador. Para se obter alta resolução (< 1ns), a frequência do *clock* de referência deve ser da ordem de GHz, exigindo a implementação de processadores muito rápidos e de alto consumo de potência. Alternativamente, podem-se usar vários contadores, sincronizados a diferentes fases do mesmo *clock*, de forma a alcançar altas resoluções com *clocks* de frequências mais baixas. A medida de tempo pode ser facilmente interpolada a partir dos resultados de todos os contadores [26].

2.2.3.3 Técnica baseada em linha-de-retardo

Com o crescente avanço na tecnologia de confecção de circuitos integrados CMOS de alta densidade, pôde-se dar início a uma nova geração de TDCs, com resolução de cerca de 100 ps e baixo consumo de potência, cuja medida de tempo é baseada em atraso de propagação de sinal em semicondutores. Nesta técnica, várias células de retardo (geralmente portas inversoras) são interligadas para formar uma linha-de-retardo digital na qual um pulso é propagado [27]. A Figura 2-7 mostra um esquema de uma destas

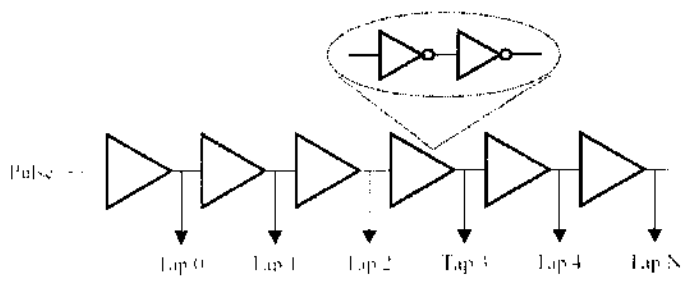

Figura 2-7: Linha-de-retardo digital, utilizando inversores duplos como célula. O retardo é obtido decodificando a palavra formada pelos sinais de saída das células [28].

linhas. A progressão do pulso ao longo da linha dá o intervalo de tempo a ser medido. Um circuito codificador, ligado ao barramento formado pelos sinais de saída das células, produz uma palavra digital de $\log_2(N)$ *bits*, onde N é o número de células da linha.

Um fato observado é que o retardo de sinais em semicondutores é extremamente dependente da temperatura e da tensão, o que restringe a sua aplicação a ambientes estáveis e controlados.

2.2.3.4 Técnica baseada em dispositivos PLL (*Phase Locked Loop*)

Os problemas relacionados com a técnica baseada em linhas-de-retardo digitais (alta sensibilidade à variação de tensão e temperatura) podem ser contornados se a linha for realimentada, formando um oscilador cuja frequência de oscilação depende do retardo total da linha. Como o retardo de cada célula varia com a sua tensão de alimentação, a linha realimentada pode ser considerada um oscilador controlado por tensão (VCO - *Voltage Controlled Oscilator*). Usando-se um circuito detector de fase pode-se calibrar a frequência de oscilação da linha a partir de uma frequência fixa [29]. A Figura 2-8 mostra um circuito com estas características, denominado de *asymmetric ring oscillator*.

Dispositivos formados por um controlador VCO e um detector de fase, configurados da forma mostrada na Figura 2-8, são conhecidos como dispositivos PLL (*Phase Locked Loop*) [30]. Circuitos PLL são amplamente utilizados em diversas áreas. Suas aplicações

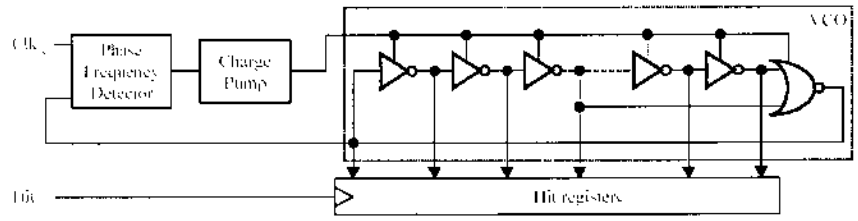

Figura 2-8: Esquema de um *asymmetric ring oscillator* - circuito PLL típico, onde o controlador VCO é uma linha-de-retardo digital realimentada [28].

mais comuns são: demodulação AM e FM, multiplicação e síntese de frequência, etc. Tais dispositivos alcançam rapidamente a estabilidade. A compensação feita pelo processo de realimentação ajusta constantemente a tensão da linha-de-retardo de modo que o atraso de propagação do sinal nas células seja sempre constante.

Podem-se conseguir grandes alcances dinâmicos, contando-se o número de oscilações na linha. Assim os *bits* menos significativos são obtidos pela codificação dos sinais da linha, e os *bits* mais significativos são provenientes do contador.

Neste tipo de circuito, a linha-de-retardo está constantemente sendo percorrida por um pulso, independentemente dos sinais de controle. Portanto, para que este circuito funcione como um TDC, são necessários, pelo menos, dois canais idênticos, um para registrar o sinal de *start* e outro para o sinal de *stop*. A palavra correspondente ao tempo entre estes dois pulsos é obtida, fazendo-se uma subtração entre estes dois dados através de um processador. Consequentemente, esta técnica permite, também, a medida de intervalos de tempos negativos.

Na *Estação de Testes,* o TDC é utilizado para converter os sinais da linha-de-retardo. O TDC escolhido é um circuito comercial cuja técnica de medida é baseada na última técnica apresentada (dispositivos PLL). O *hardware,* projetado para controle e armazenamento dos dados deste TDC, é uma adaptação da técnica de analisadores multicanais (*MultiChannel Analyser* - MCA), para medidas de tempo. O princípio de funcionamento dos MCAs é apresentado a seguir.

2.2.4 Análise dos pulsos de anodo por um MCA

O pulso de anodo, medido através do esquema mostrado na Figura 2-6, contém informação sobre a carga gerada pela interação de uma partícula com o detector (Equação 2.16). Se um grande número de pulsos, correspondendo a partículas de energia conhecida, for analisado, pode-se obter um espectro da distribuição de carga, que fornece uma medida confiável para estimativa do ganho [31].

A função de um analisador multicanal é, justamente, gerar um espectro de distribuição de uma determinada grandeza analógica, geralmente a altura do pulso do sinal a ele aplicado. Para isso, um MCA dispõe de dois componentes principais: um conversor ADC e uma memória onde a informação de distribuição é armazenada. Primeiramente, o conversor transforma a grandeza a ser medida em uma palavra digital. Depois, ocorre o processo denominado de histogramação, onde um processador decide a que canal (endereço de memória) corresponde aquela medida e incrementa o dado armazenado no canal correspondente.

No multicanal projetado para a estação, a grandeza a ser convertida é a integral do pulso. A integração é feita digitalmente, amostrando o pulso com um ADC rápido (40 MHz) e realizando somas binárias. O processo de amostragem do sinal é feito por um ADC do tipo *pipeline*.

O princípio de funcionamento dos ADCs *pipeline* é, geralmente, baseado na técnica de conversão por aproximação sucessiva [32]. Esta técnica, esquematizada na Figura 2-9, consiste em começar tentando um 1 lógico no *bit* mais significativo do conversor digital-analógico (D/A) e repetir o processo para os próximos *bits* sucessivamente. A medida que cada *bit* é testado, a saída analógica do conversor D/A é comparada com o valor do sinal a ser convertido (V_{in}). Se a saída do D/A é maior, o 1 lógico é removido daquele *bit*. No final do processo, depois que o *bit* menos significativo for testado, o código resultante do D/A é a palavra digital correspondente ao sinal de entrada V_{in}.

Para ADCs rápidos e de alta precisão, o método de aproximação sucessiva é combinado com a técnica de processamento de sinal tipo *pipeline* [32]. Nesta técnica, a conversão é dividida em estágios, onde cada estágio leva um tempo correspondente a um ciclo

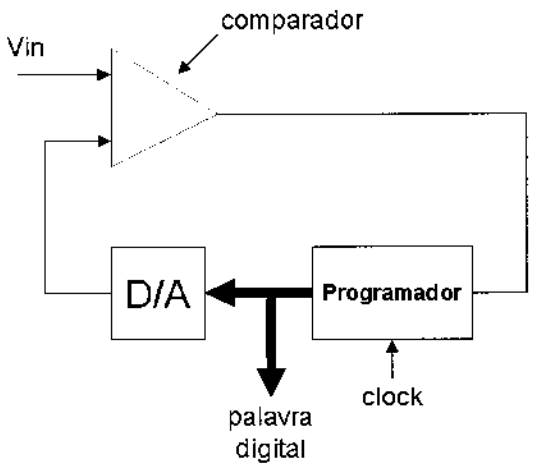

Figura 2-9: Esquema básico de conversão A/D pela técnica de aproximação sucessiva.

de *clock* para ser executado. O primeiro estágio converte *bits* mais significativos da primeira amostra do sinal e passa o resultado para o próximo estágio. Depois, durante o segundo ciclo, o primeiro estágio converte *bits* mais significativos da segunda amostra, enquanto o segundo estágio converte os próximos *bits* da primeira amostra. O resultado do segundo estágio é passado para o próximo estágio e assim por diante, até que todos os *bits* tenham sido determinados. Desta forma, o conversor libera um dado novo a cada ciclo de *clock*, mas tais ciclos podem ser feitos bem rapidamente, pois a quantidade de *bits*, a ser convertida por estágio, é bem menor. Esta técnica introduz um retardo entre o instante em que o sinal analógico foi amostrado e o instante em que seu respectivo valor digital aparece na saída do ADC, mas o aspecto de maior importância é o incremento na taxa de amostragem que esta técnica proporciona.

2.2.5 *Software* para aquisição de dados e diagnóstico

Para controlar todo o *hardware* projetado para aquisição de dados, foi desenvolvido um *software* dedicado, o *Chambers 1.0*. Além de automatizar o processo de aquisição de dados, o *Chambers 1.0* deve, também, fornecer uma interface gráfica para o usuário,

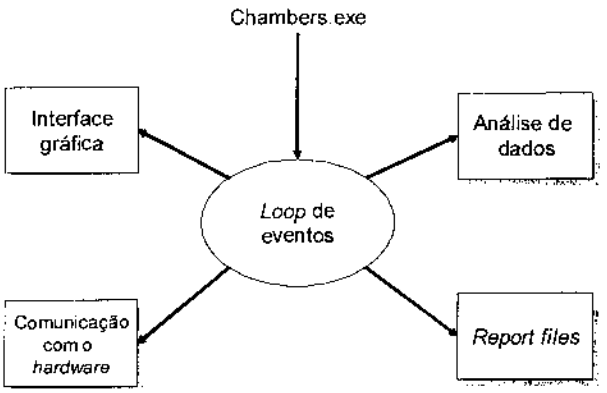

Figura 2-10: O programa *Chambers 1.0* e seus blocos principais.

fazer a análise dos dados adquiridos e gerar arquivos de *report* com os resultados finais de caracterização das câmaras. Este programa foi desenvolvido para plataforma *Windows*, utilizando o ambiente de desenvolvimento *Delphi 7.0* da *Borland* [33]. Trata-se de um programa orientado a eventos, típico do *Windows* [34] (Figura 2-10). O *Chambers 1.0* é apresentado detalhadamente no Capítulo 4.

2.3 Descrição da parte analógica

A Figura 2-11 mostra um esquema da parte analógica da estação. A cada um dos quatro planos da câmara são conectadas duas linhas-de-retardo, acopladas aos conectores de catodo, e uma barra desacopladora para o plano anódico, conforme esquematizado nas Figuras 2-6 e 2-15. Os sinais, tanto da linha-de-retardo quanto da barra desacopladora, são inicialmente pré-amplificados. A Figura 2-12 mostra um evento capturado em um osciloscópio digital para estes sinais após a pré-amplificação. O sinal na parte superior corresponde ao pulso do anodo (polaridade negativa). Os dois outros sinais são capturados nas extremidades da linha-de-retardo para o mesmo evento. O intervalo de tempo entre o pulso de anodo e o pulso de catodo determina em que ponto da linha o sinal foi injetado, ou seja, em que *pad* de catodo o evento ocorreu. Os circuitos discriminadores

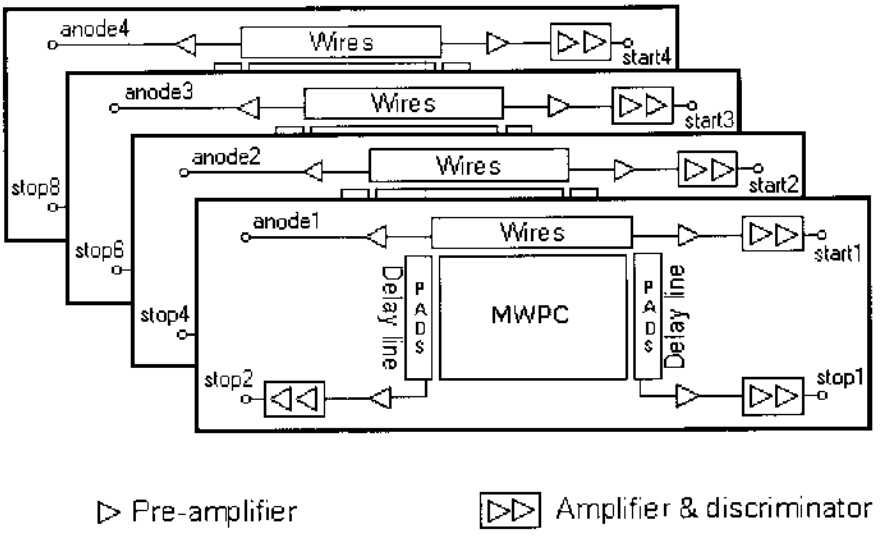

Figura 2-11: Esquema da parte analógica da *Estação de Testes*.

são os responsáveis por gerar os sinais a serem lidos pelo TDC (nível TTL), correspondentes a um evento válido. Como o sinal de anodo não passa pela linha-de-retardo, ele é usado como pulso de *start* depois da discriminação. Os sinais de catodo correspondem aos pulsos de *stop*.

Normalmente, o intervalo de tempo entre o sinal de anodo e o sinal discriminado em uma das extremidades da linha-de-retardo é suficiente para a identificação do *pad* de catodo, mas pode-se também utilizar a diferença de tempo entre sinais das duas extremidades da linha. Isto melhora a resolução e diminui o *background* no histograma, gerado por ruído no discriminador. Estes dois modos de aquisição são tratados no capítulo que descreve o *hardware* responsável pelo processo de histogramação (Capítulo 4).

É necessário também enviar o sinal de anodo ao analisador multicanal, onde ele é integrado, para que se obtenha uma informação referente à carga gerada no evento. Em seguida, é feita a histogramação dos valores obtidos na integração.

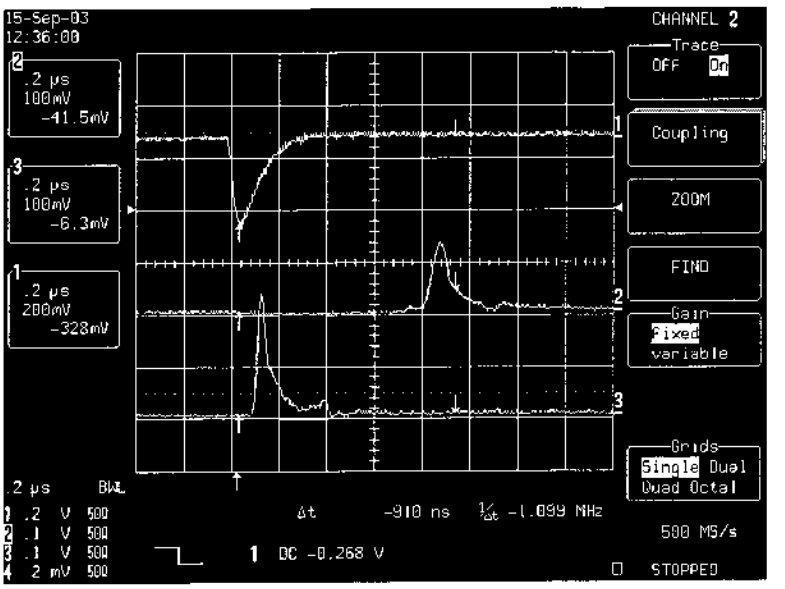

Figura 2-12: Sinais de anodo e catodo capturados por um osciloscópio digital após os pré-amplificadores. Os sinais de catodo (positivos) são coletados nas extremidades da linha-de-retardo e correspondem ao mesmo evento.

2.3.1 Projeto da linha-de-retardo

O dimensionamento dos componentes da linha-de-retardo deve ser feito levando-se em conta a resolução temporal da câmara. Tal resolução pode ser estimada com a ajuda do aparato experimental mostrado na Figura 2-13. Dois detectores, com resolução temporal muito menor do que a da câmara (por exemplo, cintiladores rápidos), são posicionados na parte superior e inferior da mesma. A coincidência entre os disparos dos dois cintiladores, devido a um evento, é utilizada como *start* para um TDC. O sinal de disparo da câmara, atrasado por um retardo fixo t_0, é utilizado como sinal de *stop*. Se um grande número de medidas for feita, o espectro temporal resulta em uma distribuição normal centrada em t_0. A largura desta distribuição (δ) é a resolução temporal da câmara.

Como especificado pelo procedimento de *trigger* do *Sistema de Múons*, cinco câmaras, uma de cada estação (M1 a M5), devem responder a um evento dentro de um intervalo de tempo de 25 ns. Portanto, para que o *Sistema de trigger* possa identificar 5 eventos

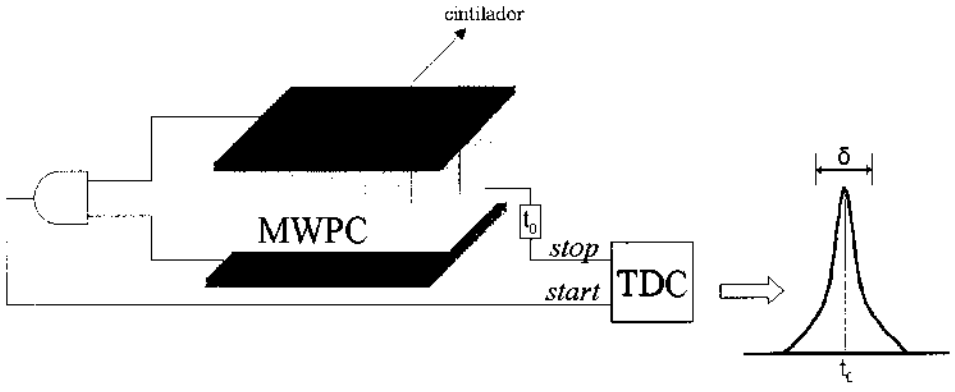

Figura 2-13: Aparato experimental para medida da resolução temporal da câmara.

distintos dentro deste intervalo, a resolução temporal de cada câmara não pode ser maior que 5 ns. De fato, medidas realizadas em *test-beams* mostraram que este parâmetro é da ordem de 4 ns [23].

Para que cada *pad* possa ser identificado, com boa resolução, por uma medida de intervalo de tempo, a linha-de-retardo deve ser projetada de forma que o retardo entre *pads* seja bem maior que a resolução temporal da câmara ($\gg$ 5 ns). A Figura 2-14 apresenta dois gráficos que mostram a separação, no tempo, entre sinais tomados em *pads* conectados em células consecutivas de duas diferentes linhas-de-retardo. Os pulsos pontilhados mostram que a resolução temporal impõe uma imprecisão intrínseca na medida de tempo, correspondente ao sinal injetado em um *pad*. Em consequência, o retardo entre *pads* (t_0) deve ser $\gg \delta$ para que o evento seja identificado inequivocadamente.

Na prática, é suficiente um retardo entre *pads* 5 vezes maior que a resolução temporal da câmara, ou seja, 25 ns. Contudo, pela Equação 2.12, uma célula com este retardo resultaria em um produto LC relativamente alto, implicando em uma banda passante limitada (Equações 2.2 e 2.3). Optou-se, então, por utilizar três células de 8 ns cada, acopladas entre *pads*, conforme mostrado na Figura 2-15. Esta figura mostra também, que, no local onde os *pads* são conectados à linha, o capacitor é substituído pela própria capacitância do *pad* (representados por capacitores na cor cinza). Um mapeamento desta

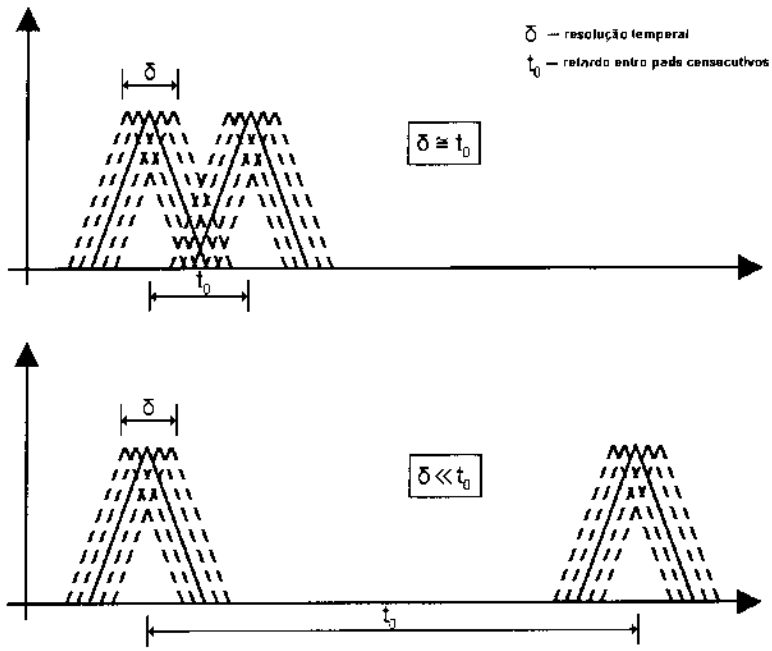

Figura 2-14: Comparação entre resolução temporal e retardo entre células para duas linhas-de-retardo. Na linha onde $\delta \ll t_0$, o *pad* é identificado com maior precisão.

capacitância feito em um dos planos de uma câmara protótipo mostra que o valor médio desta capacitância é de 40 pF (Figura 2-16). Este deve ser, portanto, o valor escolhido para os capacitores da linha. Consequentemente, os indutores devem ser de 1.6 uH (Equação 2.12), o que resulta em uma impedância de 200 Ω (Equação 2.1). Uma outra vantagem de se usar três células entre *pads* é que, desta forma, a variação da capacitância de um *pad* para outro não afeta a linearidade da linha de forma expressiva.

Com relação ao *layout*, as linhas-de-retardo foram projetadas de forma a se acoplarem aos conectores dos *pads* de catodo nas laterais da câmara. A Figura 2-17 é uma foto da linha-de-retardo projetada para a *Estação de Testes*.

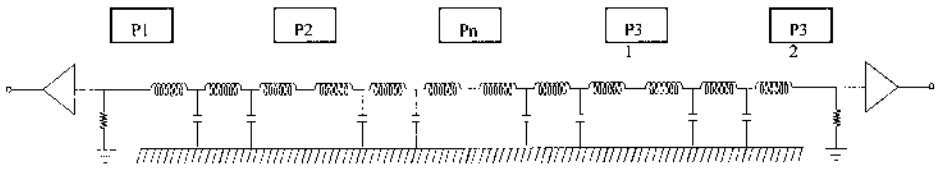

Figura 2-15: Esquema de conexão dos *pads* de catodo à linha-de-retardo projetada para a *Estação de Testes*.

48.0	47.6	46.1	45.7
47.0	46.9	45.2	44.8
41.8	41.8	40.5	40.5
37.4	37.4	36.7	36.7
47.6	47.4	46.1	45.5
46.3	46.6	45.3	44.5
41.1	41.6	40.2	40.0
36.9	37.1	36.2	36.4

Figura 2-16: Mapeamento da capacitância dos *pads* (em pF) de catodo em um dos planos de uma câmara protótipo.

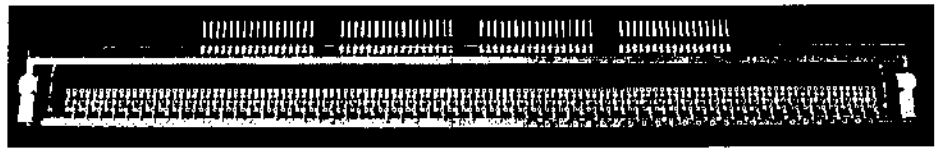

Figura 2-17: Foto da linha-de-retardo projetada para a *Estação de Testes*.

2.3.2 Barra desacopladora do Anodo

A função da barra desacopladora é extrair o pulso de tensão gerado no anodo pelo processo de avalanche de elétrons gerado pela ionização de moléculas do gás. Para isso a barra curto-circuita todos os fios de anodo de um plano e desacopla o pulso do nível de alta tensão, através de um capacitor (Figura 2-6). A barra desacopladora é geometricamente projetada para ser encaixada nos conectores de anodo, situados na parte inferior da câmara (Figura 1-8 b).

2.3.3 Pré-amplificador

Os pré-amplificadores são utilizados para transformar os pulsos de anodo e de catodo em sinais de amplitudes mensuráveis para o circuito discriminador e para o MCA. Este circuito foi projetado para ter ganho 100 e a sua banda passante deve ser tal que possa amplificar os sinais da linha-de-retardo e da barra desacopladora sem distorcê-los criticamente. Testes com um gerador de pulso mostraram um *risetime* de 50 ns, o que é compatível com estes requisitos. O esquemático do pré-amplificador projetado se encontra no Apêndice A.

2.3.4 Amplificador e discriminador

Um circuito de discriminação, utilizando o comparador NE529, foi projetado para gerar os sinais que ativam o TDC. Antes do discriminador, tais sinais passam por um amplificador de ganho 10 baseado no circuito integrado LM733. A Figura 2-18 mostra o circuito, projetado com cinco canais de amplificação e discriminação. Um dos canais dispara em transição de descida, sendo, portanto, dedicado a discriminar o sinal de anodo. Os outros quatro canais são utilizados para os sinais provenientes da linha-de-retardo (sinais de catodo). O Apêndice A apresenta o esquemático completo deste circuito.

Figura 2-18: Foto do circuito amplificador-discriminador de 5 canais projetado para a *Estação de Testes*.

2.4 Módulo de Processamento de Dados

A Figura 2-19 mostra um esquema da parte de processamento dos dados da estação, denominada *Módulo de Processamento de Dados* (MPD). Os sinais já discriminados dos quatro anodos são usados independentemente ou podem ser combinados para formar o pulso de *start* do TDC. Este estágio é simplesmente um AND ou um OR lógico, caso se queira ativar o TDC quando houver eventos simultâneos nos quatros planos ou em qualquer um deles, respectivamente. Ao todo são necessários 8 canais independentes de TDC (*stop*), já que a estação usa 8 linhas-de-retardo.

O *Módulo TDC* converte e processa os dados da linha-de-retardo e gera 8 histogramas em uma memória local, um para cada linha. Todo o processo é controlado por uma FPGA (*Field Programmable Gate Array*). No MCA, assim como no módulo TDC, uma FPGA controla o processo de histogramação em uma memória local.

A interface com o computador é feita através da porta-paralela, por ser uma interface de fácil implementação, presente em praticamente todos os PCs (*Personal Computers*). Embora esta interface apresente baixa velocidade de acesso, todo o processamento é feito por *hardware* no MPD. A porta-paralela é utilizada apenas para ler os dados já histogramados, ou seja, sua velocidade não influencia na taxa de aquisição de dados.

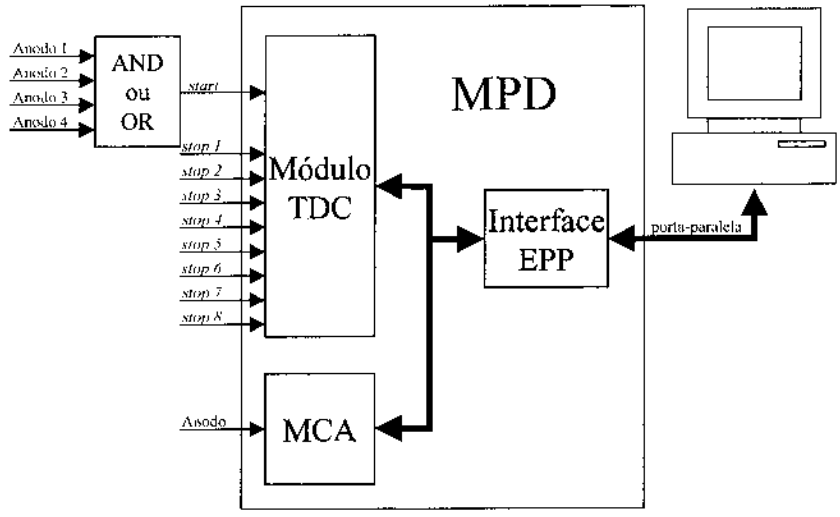

Figura 2-19: *Módulo de Processamento de Dados* (MPD) e seus três blocos principais.

De acordo com a especificação *IEEE Std 1284* existem cinco padrões de transferência de dados pela porta-paralela [35]. Dentre eles, o padrão EPP (*Enhanced Parallel Port*) foi o escolhido, porque permite transferência bidirecional de dados a uma taxa de até 2 MBytes por segundo, atendendo às necessidades da *Estação de Testes*. O bloco *Interface EPP* do MPD foi projetado para adequar as características elétricas do MPD (lógica TTL) às da porta EPP, estando de acordo com a especificação *IEEE Std 1284*.

Devido a sua complexidade e importância, o MPD será abordado em mais detalhes nos dois próximos capítulos. A Figura 2-20 apresenta uma foto da *Estação de Testes* completa, conectada a uma câmara de múons do LHCb.

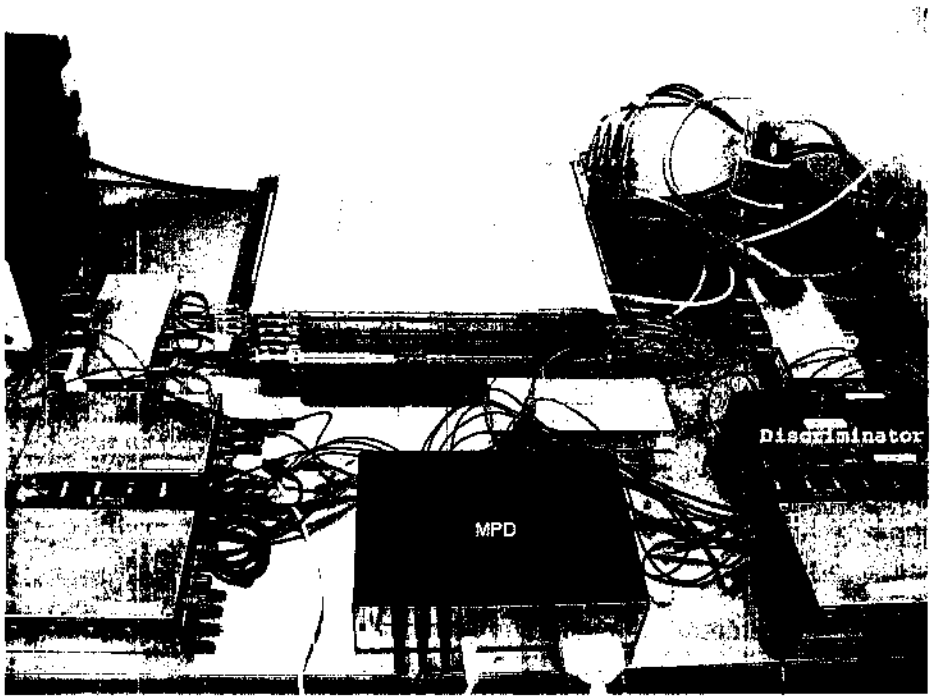

Figura 2-20: Foto de uma câmara de múons conectada à *Estação de Testes*.

Capítulo 3

Módulo de Processamento de Dados

O *Módulo de Processamento de Dados* (MPD) é formado por três submódulos (Figura 2-19):

1. **Módulo TDC** - composto por oito canais de conversão tempo-digital e circuitos para histogramação. É utilizado para a análise da eficiência relativa e identificação de *pads* defeituosos. Utiliza informação de atraso de propagação de sinal nas linhas-de-retardo para identificação dos *pads* de catodo.

2. **Analisador Multicanal (MCA)** - seus componentes principais são: um ADC do tipo *pipeline*, um circuito integrador digital e um circuito histogramador. Sua função é estimar os parâmetros de ganho global e uniformidade de ganho da câmara a partir da integral do sinal de anodo. O histogramador utiliza esta informação para gerar um espectro de energia em uma memória local.

3. **Interface EPP** - faz a comunicação do *Módulo TDC* e do *MCA* com a porta-paralela do PC. Sua função é adequar os parâmetros elétricos dos sinais do barramento local à especificação do padrão EPP (*Enhanced Parallel Port*).

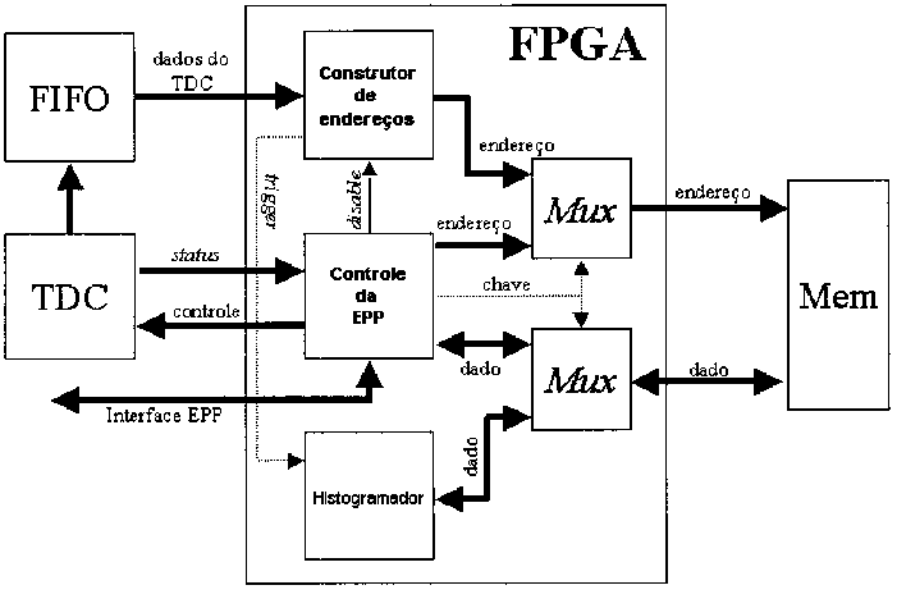

Figura 3-1: Diagrama em blocos do *Módulo TDC*.

3.1 Descrição do *hardware*

O *Módulo TDC* pode ser melhor entendido com o auxílio do diagrama em blocos da Figura 3-1, onde alguns sinais foram omitidos em favor da clareza. Estes sinais serão apresentados tão logo se fizerem necessários, com o detalhamento de cada bloco.

Todo o controle do *hardware* é feito pelo bloco de *Controle da EPP* através de comandos da *Interface EPP* via porta-paralela do PC, que, por sua vez, recebe comandos do programa de interface com o usuário. O bloco de *Controle da EPP* é responsável, por exemplo, por configurar o TDC e enviar seus dados de *status* ao programa. Este bloco também compartilha o acesso à memória local com os blocos *Construtor de Endereços* e *Histogramador*. O *Construtor de Endereços* é o bloco responsável por interpretar os dados do TDC e gerar o endereço de memória cujo dado será incrementado pelo bloco *Histogramador*. O sinal *disable* desabilita o bloco *Construtor de Endereços* quando o PC necessita acessar a memória através do bloco de *Controle da EPP*. Para evitar que dados, vindos do TDC, sejam perdidos durante este processo, uma FIFO é utilizada para armazenar

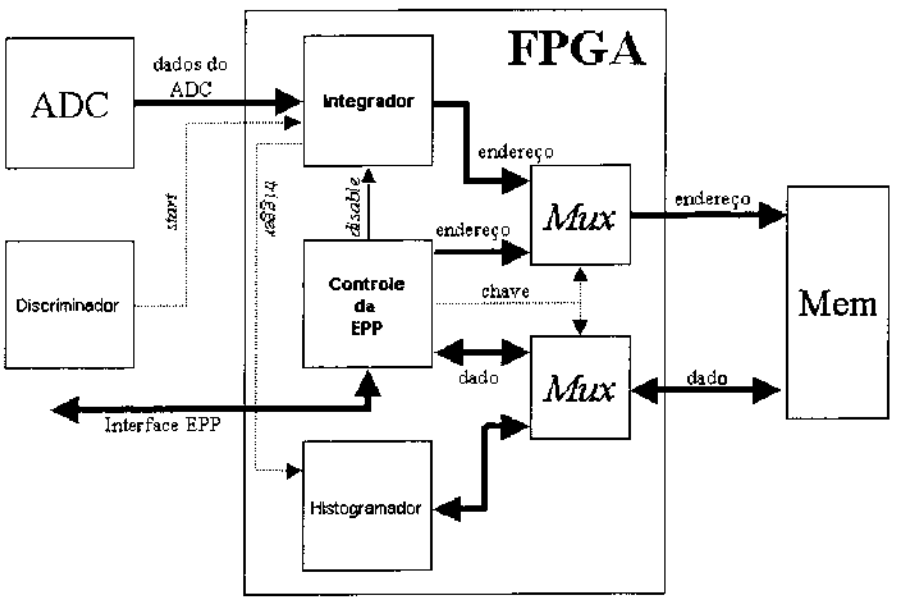

Figura 3-2: Diagrama em blocos do MCA.

temporariamente as últimas conversões. Quando o bloco *Construtor de Endereços* está habilitado, ele dispara um sinal de *trigger*, acionando o *Histogramador*, assim que um novo endereço de memória estiver disponível para a histogramação. Note-se que os circuitos de controle, de construção de endereços e de histogramação são implementados em uma única FPGA.

A mesma filosofia é utilizada no desenvolvimento do MCA (Figura 3-2), onde um bloco de *Controle da EPP* é responsável por controlar o sistema de conversão - que, neste caso, é um ADC *pipeline* - e arbitrar o acesso à memória. No MCA, o bloco *Construtor de Endereços* é mais apropriadamente denominado de bloco *Integrador*, que, como o próprio nome sugere, fornece, como endereço de memória, a integral do pulso digitalizado. Para que esse bloco comece o processo de integração, um circuito discriminador dispara um sinal de *start*, informando a presença de um pulso válido.

Finalmente, o terceiro bloco do MPD, o *Interface EPP*, é composto por um único circuito integrado, o *SN74LV161284* da *Texas Instruments* [36]. Este circuito funciona como uma ponte, fazendo uma interface entre os níveis de sinais do padrão EPP (*Enhanced*

Palellel Port), dados pela especificação IEEE Std 1284 [35], e a lógica TTL (*Transistor Transistor Logic*) das FPGAs. O *Módulo TDC* e o *MCA* são ligados ao bloco *Interface EPP* em paralelo. Para que o programa saiba com qual dos dois deve se comunicar, foi desenvolvido um protocolo de endereçamento, implementado dentro de cada bloco de *Controle da EPP*.

3.1.1 O conversor TDC

O bloco TDC da Figura 3-1 é composto por um circuito integrado conversor tempo-digital, e por toda a eletrônica necessária ao seu funcionamento. O conversor escolhido foi o TDC F1 [37], desenvolvido pela empresa alemã *Acam* [38] em colaboração com a Faculdade de Física da Universidade de Freiburg, para uso no experimento COMPASS [39], no CERN.

O F1 tem 8 canais com resolução em torno de 120 ps por canal e 16 *bits* de profundidade. A resolução é ajustável por *software* e calibrada por um cristal de quartzo através de um circuito PLL (*Phase Locked Loop*) (§ 2.2.3.4). Há apenas um sinal de *start* comum a todos os canais. Pelo modo como o F1 é configurado neste trabalho, a taxa máxima de conversão pode chegar a 2 milhões de conversões por segundo (2 MHz).

Os sinais de entrada (*start* e *stop*) seguem o padrão diferencial LVPECL (*Low Voltage Pseudo ECL*), mas podem atuar como comparadores, bastando aplicar um limiar de disparo à entrada inversora. Para simplificar e tornar mais poderoso o uso desse nível de *threshold*, o F1 oferece uma interface para controlar diretamente alguns conversores DAC (*Digital-to-Analogic Converters*) da *Analog Devices* [40]. Deste modo, os níveis de *threshold* podem ser controlados por *software* através de registradores de configuração do F1.

O TDC F1 é controlado através de escrita em seus registradores de configuração. Este procedimento é feito de forma serial, utilizando um protocolo específico. O bloco de *Controle da EPP* do *Módulo TDC* recebe os comandos do programa *Chambers 1.0* e gera os sinais necessários para a configuração dos registradores do F1.

A Figura 3-3 é um esquema do bloco TDC. O sinal de *start* e os 8 sinais de *stop*

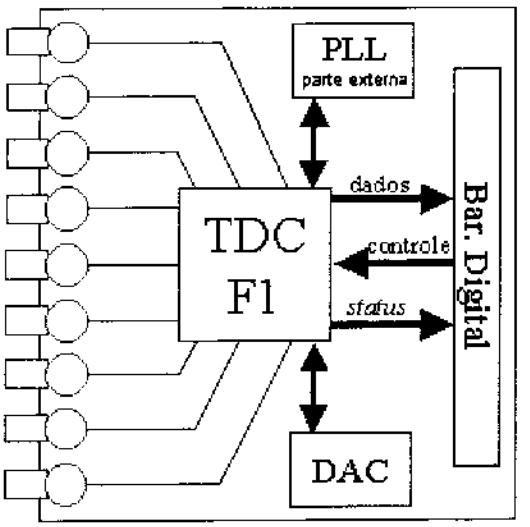

Figura 3-3: Esquema do circuito impresso que contém o TDC F1.

chegam ao módulo através de cabos coaxiais de impedância 50 Ω e conectores padrão LEMO. O barramento digital contém todos os sinais de dados e configuração, além de alguns sinais de *status* do F1. O circuito PLL, que ajusta a resolução do TDC, requer uma parte externa ao F1, feita com resistores e o regulador de tensão ajustável LM317 [41]. O bloco ainda conta com um conversor DAC de 8 canais, o DAC8841 [42], para controle dos níveis de *threshold* dos sinais de entrada, via *software*.

Para preservar a modularidade, optou-se por desenvolver um PCB independente para este bloco. A Figura 3-4 mostra o resultado. O esquemático completo deste circuito é apresentado no Apêndice A.

3.1.2 A interface histogramadora do Módulo TDC

Um outro circuito impresso foi projetado para acomodar os blocos restantes do *Módulo TDC* (Figura 3-1), com exceção do bloco que contém o TDC F1 já explicado no item anterior. A Figura 3-5 mostra um esquema deste circuito de interface. O esquemático completo se encontra no Apêndice A.

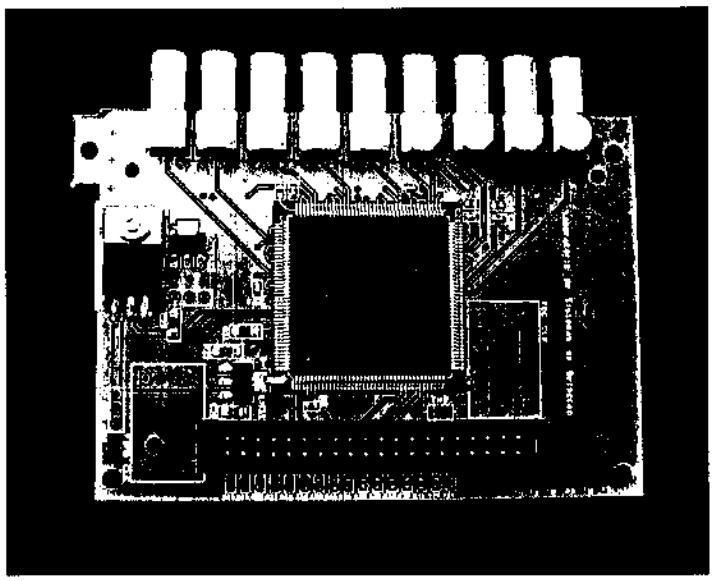

Figura 3-4: Foto do PCB que contém o TDC F1.

O barramento digital faz a comunicação com o conversor TDC. Todos os sinais deste barramento se conectam à FPGA. Os sinais de dados passam primeiramente por uma FIFO síncrona. A escrita, na FIFO, é controlada pelo TDC F1, e a leitura, pelo bloco *Construtor de Endereços* na FPGA. Como a taxa máxima de conversões do F1, nesse modo de configuração, é de 2 MHz, escolheu-se um *clock* de leitura da FIFO de 20 MHz, ou seja, uma ordem de grandeza superior, para que a FIFO seja esvaziada rapidamente, assim que a memória for liberada novamente para histogramação.

A FPGA escolhida foi a XC4010E [43] por ter capacidade e número de pinos suficientes, além de apresentar tempos de propagação que não afetam o desempenho do sistema.Uma EEPROM é utilizada para armazenar o código de configuração desta FPGA.

O circuito de comunicação com a porta EPP, o SN74161284, e um conector de porta-paralela DB-25, foram também acrescentados a este *layout,* evitando-se o projeto de mais uma interface para somente estas duas peças. A Figura 3-6 mostra o produto final.

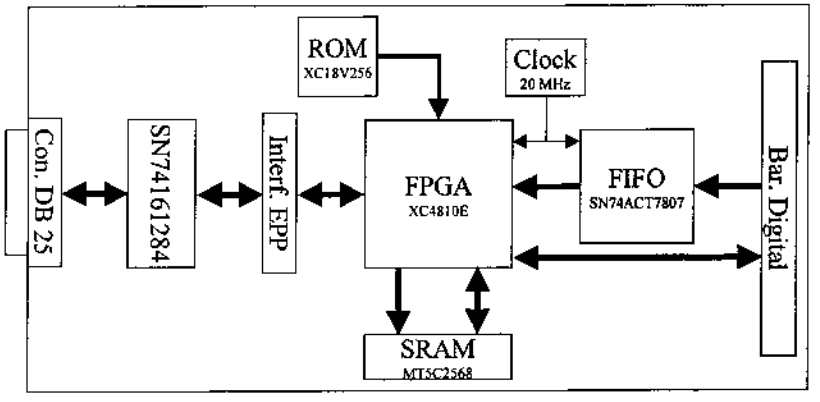

Figura 3-5: Esquema da interface histogramadora do *Módulo TDC*.

3.1.3 O conversor ADC

Este módulo é composto por um conversor analógico-digital do tipo *pipeline* e um circuito discriminador. O conversor usado foi o ADS828 [44]. Este ADC tem resolução de 10 *bits* e amostra sinais de 2 Vpp a uma frequência de 40 MHz. A Figura 3-7 mostra um esquema simplificado desta interface. De acordo com as especificações, o ADS828 converte sinais de 1.5V a 3.5V, portanto, faz-se necessário o uso de um circuito somador analógico que adiciona uma tensão fixa de 2.5V ao sinal de entrada. Este somador foi implementado com o amplificador operacional OPA690 [45]. O sinal de disparo do discriminador, os 10 *bits* de dados e alguns sinais de controle do ADC formam o barramento digital que se comunica com a interface histogramadora. O Apêndice A contém o esquemático completo deste circuito.

3.1.4 A interface histogramadora do MCA

Esta interface segue o mesmo princípio de funcionamento da interface histogramadora do TDC (§ 3.1.2). A Figura 3-8 mostra uma foto dos circuitos projetados para o conversor ADC e sua interface histogramadora.

Os circuitos, mostrados nas Figuras 3-4, 3-6 e 3-8, formam a parte física do MPD que foram fixados e interconectados em uma caixa metálica (Figura 3-9).

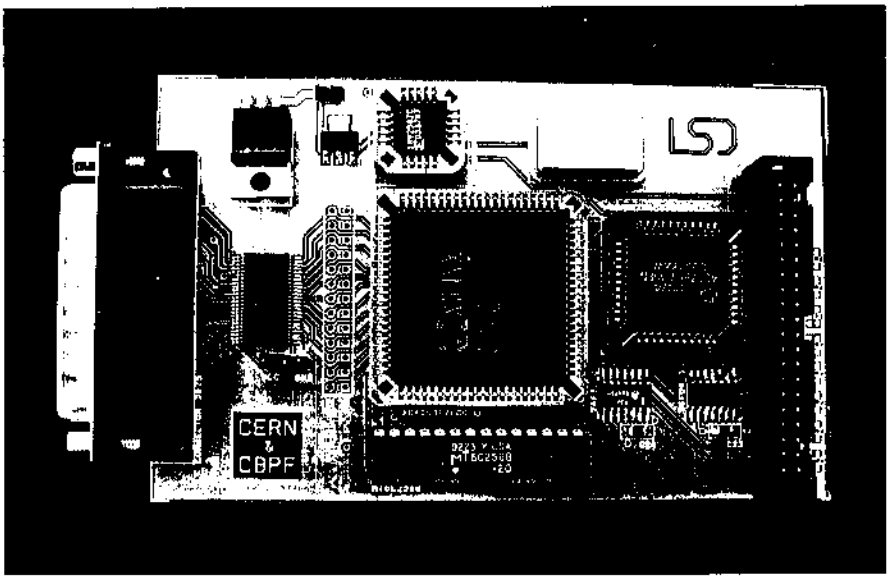

Figura 3-6: Foto da interface histogramadora do *Módulo TDC.*

3.2 FPGAs

FPGA (*Field Programable Gate Array*) é um tipo de dispositivo lógico programável de alta densidade e de propósito geral, para a implementação de circuitos digitais.Ele contém um conjunto de elementos de circuitos lógicos que podem ser interligados de diferentes maneiras para executar uma função específica [46].

Neste projeto, a maior parte do código das FPGAs foi escrita em VHDL (*Very-high-speed-integrated-circuit HDL*). VHDL [46] é uma linguagem de descrição de *hardware* (HDL - *Hardware Description Language*). HDL é similar a uma linguagem de programação de alto nível, porém é usada especificamente para descrever *hardware*, ao invés de ser executada por um processador.

3.2.1 FPGA do Módulo TDC

A Figura 3-1 mostra os blocos internos da FPGA do *Módulo TDC.* Dentre estes, os blocos de *Controle da EPP*, *Construtor de Endereços* e *Histogramador* foram escritos em

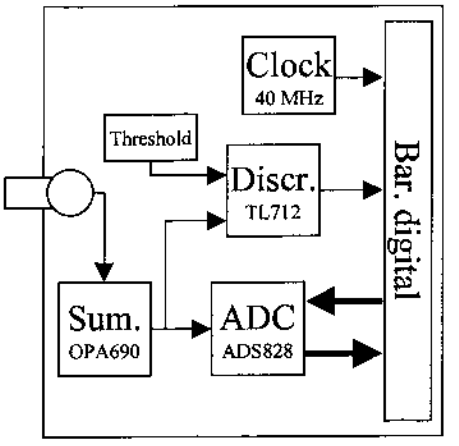

Figura 3-7: Esquema simplificado do módulo de conversão A/D do MCA.

VHDL. A ligação entre estes blocos, bem como o desenvolvimento dos blocos restantes, foi feita na forma de esquemático. O Apêndice B mostra este esquemático por completo.

3.2.1.1 Bloco de Controle da EPP

O Bloco de *Controle da EPP* recebe comandos do programa de interface com o usuário, via porta-paralela. Tal porta deve estar configurada em modo EPP para que a comunicação funcione corretamente. Antes de descrever o funcionamento deste bloco, convém apresentar um breve esclarecimento do protocolo de comunicação do padrão EPP.

A porta-paralela foi originalmente projetada para conectar o PC às impressoras. Seu desempenho era de, no máximo, 200 KB/s. Esta baixa velocidade se devia ao fato de que a porta-paralela era orientada por *software*, sendo necessárias várias instruções de I/O (códigos gerados pelo programa, para que o computador acesse o *hardware*) para negociar a transferência de um único *byte* para a impressora. Este processo é denominado de *handshake* por *software*. A leitura de dados era ainda mais lenta, pois existiam apenas 4 *bits* para receber os dados. Em março de 1994 surgiu a recomendação IEEE 1284 (*Standard Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers*) [35] que deu origem a novos padrões denominados EPP (*Enhanced Parallel*

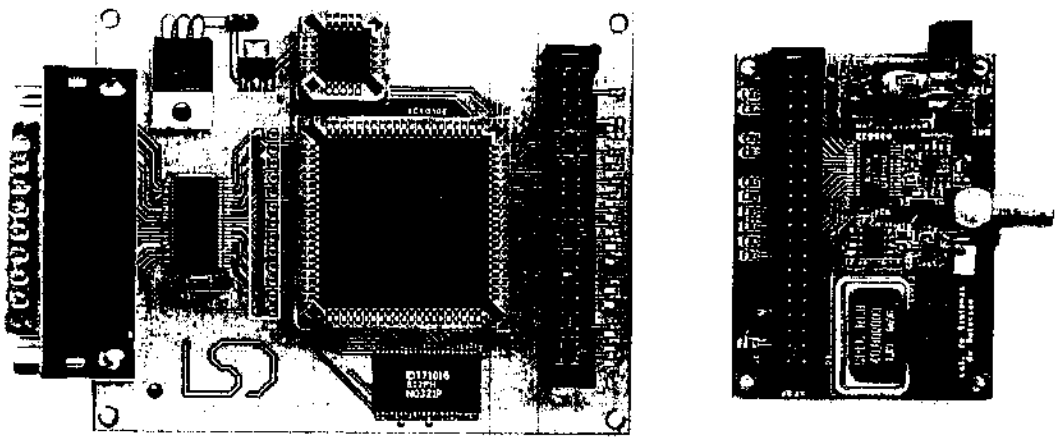

Figura 3-8: Foto das duas interfaces que compõem o MCA. O conversor ADC (à direita) e a interface histogramadora (à esquerda).

Port) e ECP (*Extended Capability Port*). Esta recomendação modifica a função de alguns pinos desta interface, adicionando recursos para transmissão bidirecional de dados, com *handshake* elaborado por *hardware*, permitindo a transferência de dados, com uma única instrução de I/O, a uma taxa máxima de 2 MB/s. O protocolo de transferência de dados pela porta EPP é explicado, com mais detalhes, em [47].

As principais funções do bloco de *Controle da EPP* são:

1. **Fazer o *handshake* com a porta EPP**

   O *handshake*, neste caso, é uma lógica combinacional simples (ver Apêndice B).

2. **Gerar sinais de controle para acesso à memória local**

   Quando o programa necessita acessar a memória local, o bloco de *Controle da EPP* gera os sinais de escrita e leitura necessários, e liga o barramento de dados da memória à porta-paralela. A memória é acessada linearmente pelo programa, ou seja, o bloco de *Controle da EPP* é responsável por incrementar automaticamente

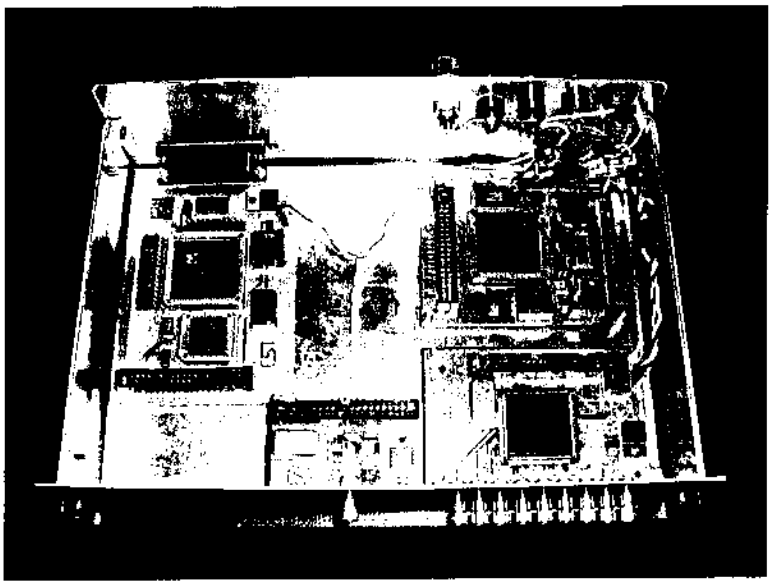

Figura 3-9: Parte física do MPD, com as quatro interfaces que a compõem. Os cabos de comunicação entre as interfaces foram retirados, por questão de clareza.

o endereço da memória antes de cada acesso feito a ela pelo programa. Isto faz com que o acesso à memória seja mais rápido e transparente ao programa de aquisição.

3. **Gerar sinais de controle para a aquisição de dados**

   Os sinais de controle para a aquisição de dados são enviados pelo programa e armazenados em registradores, dentro do bloco de *Controle da EPP*. Dentre eles os mais importantes são: os sinais de configuração serial do TDC e o decodificador de endereços, que identifica se o programa deve interagir com o *Módulo TDC* ou com o *MCA*.

O código completo, em VHDL, do bloco de *Controle da EPP* é fornecido no Apêndice B, onde comentários ajudam a identificar cada função.

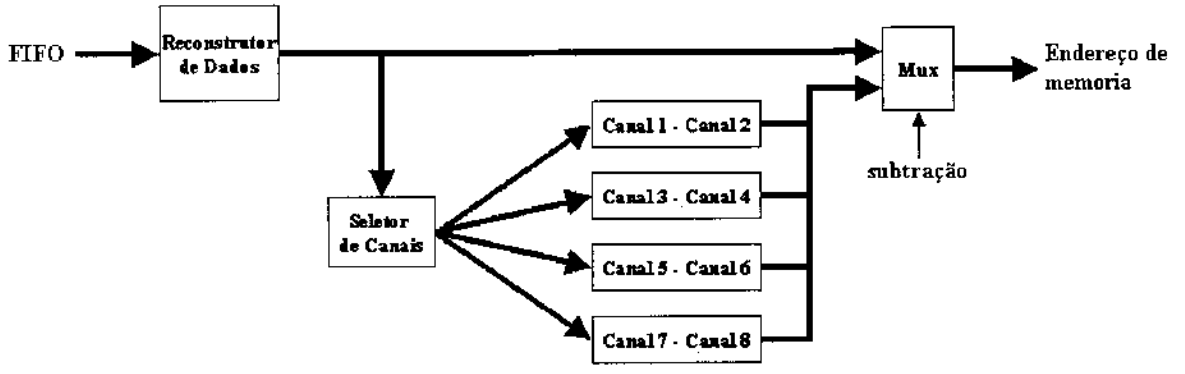

Figura 3-10: Sub-blocos do bloco *Construtor de Endereços.*

3.2.1.2 Bloco Construtor de Endereços

A função do bloco *Construtor de Endereços* é gerar um endereço de memória a partir da palavra de conversão do TDC. O F1 pode enviar os 24 *bits* deste barramento em paralelo, ou pode-se optar por um barramento de 8 *bits*, necessitando-se assim três leituras consecutivas para se ter a informação completa. A segunda opção foi a escolhida, por diminuir o número de trilhas, facilitando o projeto do *layout*, apesar de aumentar a complexidade do bloco *Construtor de Endereços*. Para uma melhor compreensão de sua funcionalidade, este bloco é dividido em sub-blocos (Figura 3-10).

O *Reconstrutor de Dados*, remonta os dados de 24 *bits* do TDC e descarta alguns *bits* desnecessários. Caso se queira combinar a informação das duas extremidades da linha-de-retardo, uma subtração deve ser realizada entre estas informações (§ 2.3). Assim, uma linha deve ser conectada aos canais 1 e 2 do TDC, uma outra linha aos canais 3 e 4, e assim por diante, para que a subtração seja feita corretamente. Um *flag* vindo do bloco de *Controle da EPP*, controlado pelo programa *Chambers 1.0*, arbitra se os dados devem ser subtraídos ou passados diretamente ao endereço de memória. Para saber para qual bloco de subtração um dado deve seguir, o *Seletor de Canais* o direciona para um dos *Subtratores*.

Alguns destes sub-blocos foram desenvolvidos como *Máquinas de Estados*. Máquinas

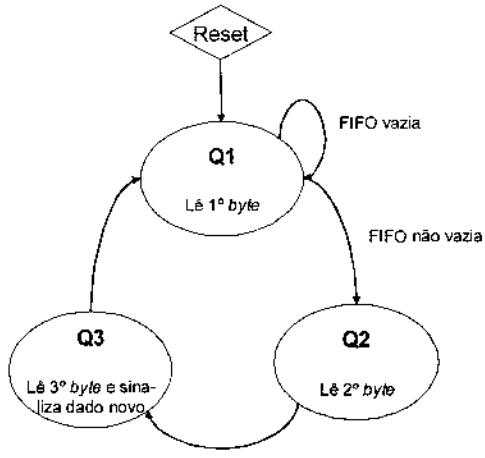

Figura 3-11: Diagrama de estados do *Reconstrutor de Dados*.

de estados são circuitos seqüenciais, cuja saída e estado atuais dependem de seu comportamento anterior, e dos valores de entrada atuais [46].Uma das maneiras de se representar uma máquina de estados é um gráfico, denominado de *Diagrama de Estados*, onde os estados são representados por círculos, interligados por setas, que representam o fluxo entre estados. A vantagem de se trabalhar com diagramas de estados é que eles são muito intuitivos e dão um maior grau de abstração ao desenvolvimento de circuitos digitais sequenciais. Além disso, uma vez que eles tenham sido projetados corretamente, é relativamente simples transformá-los em um código VHDL [46].

3.2.1.2.1 Reconstrutor de Dados O diagrama de estados da Figura 3-11 descreve o *Reconstrutor de Dados*. Alguns sinais auxiliares, como, por exemplo, sinais de controle da FIFO, foram omitidos para facilitar a explicação. O Apêndice B mostra esta máquina completa descrita em VHDL. Assim como em todas as máquinas desenvolvidas neste projeto, um sinal assíncrono de *reset* é responsável por inicializar as máquinas no estado Q1. Este sinal é proveniente do bloco de *Controle da EPP* e é gerado pelo programa de aquisição.

A máquina permanece no estado Q1 enquanto a FIFO estiver vazia. Cada estado é responsável por analisar 8 *bits* (1 *byte*) dos 24 *bits* de dados do TDC, e armazená-los em

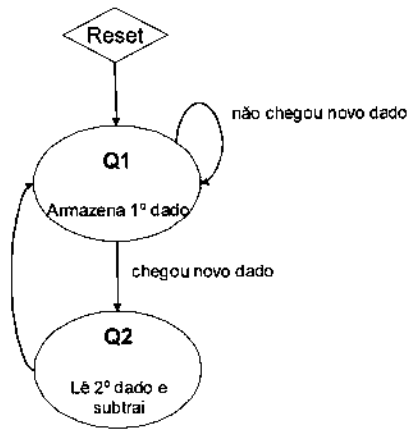

Figura 3-12: Diagrama de estados dos *Subtratores*.

posições pré definidas na palavra final que compõe o endereço de memória correspondente. Em cada estado alguns *bits* são descartados. Os três bits menos significativos do dado de conversão do TDC, por exemplo, não são utilizados. Com isso a resolução aumenta de 120 ps para cerca de 1 ns, valor compatível com a resolução temporal de 4 ns da câmara (§ 2.3.1). Quando a máquina atinge o estado Q3, ela dispara um sinal de sincronismo, avisando ao *Seletor de Canais* que um novo dado está pronto para ser processado.

3.2.1.2.2 Seletor de Canais O *Seletor de Canais* contém uma lógica combinacional simples que redireciona o sinal de sincronismo do *Reconstrutor de Dados* para uma das máquinas subtratoras, dependendo de qual canal do TDC foi utilizado na medida.

3.2.1.2.3 Subtratores A máquina de subtração (Figura 3-12) apresenta apenas dois estados. O estado Q1 aguarda a chegada do primeiro dado e o armazena. Em Q2 é feita a subtração com o segundo dado e a máquina é reiniciada.

3.2.1.3 Histogramador

A função do *Histogramador* é incrementar o dado da memória no endereço correspondente gerado pelo bloco *Construtor de Endereços*. O processo é, normalmente, executado em

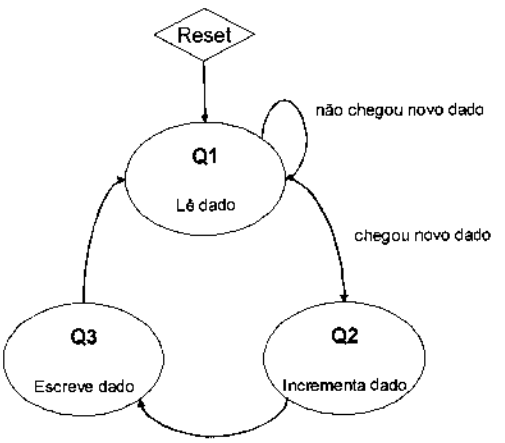

Figura 3-13: Diagrama de estados do *Histogramador*.

três etapas (Figura 3-13): ler o dado na memória, incrementá-lo, e escrever o novo valor no mesmo endereço. Esta máquina aguarda o sinal de sincronismo do *Construtor de Endereços* para iniciar o processo de histogramação. O Apêndice B mostra o código completo do *Histogramador*.

3.2.2 FPGA do analisador multicanal

A principal diferença entre a FPGA do *Módulo TDC* e a FPGA do analisador multicanal está no bloco *Construtor de Endereços*. No MCA este bloco deve acumular os dados vindos do ADC durante uma janela de tempo (§ 2.2.4) especificada pelo bloco de *Controle do EPP*. O processo de integração tem início com um sinal de disparo do discriminador. O valor resultante é utilizado como um endereço de memória para o bloco *Histogramador*.

A Figura 3-14 mostra a máquina de estados do *Integrador*. O estado Q1 simplesmente aguarda o disparo do discriminador. Em Q2 é calculada a linha de base do sinal discriminado. Isto é possível graças ao atraso imposto pelo ADC *pipeline* entre o momento em que o sinal é amostrado e o instante em que seu respectivo valor digital aparece na saída do ADC, fazendo com que a máquina tenha acesso a amostras do sinal antes do instante de disparo do discriminador. O estágio Q3 faz a integração propriamente dita,

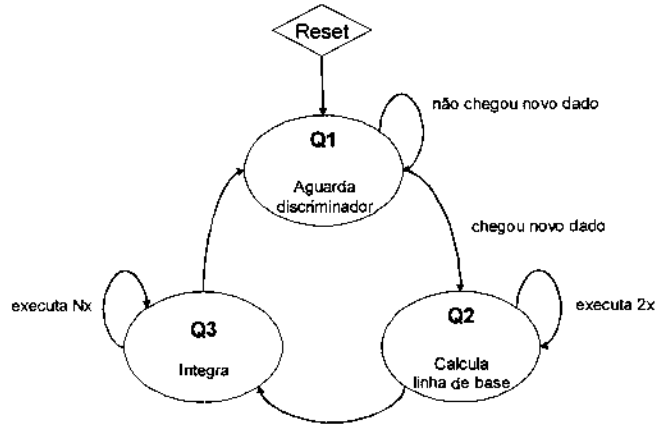

Figura 3-14: Máquina de estado do *Integrador*.

acumulando N amostras do sinal de entrada. O valor de N é determinado pelo programa. O código completo do *Integrador* está no Apêndice B.

Uma vantagem deste processo de integração quando comparado a processos de integração analógicos, como a utilização de um pré-de-carga ou um ADC de carga, é que, na integração digital, a linha de base pode ser suprimida do resultado da integral, de modo que ela não influencie no espectro de distribuição de carga resultante.

Capítulo 4

O programa Chambers 1.0

O programa desenvolvido como parte do *Módulo de Processamento de Dados* da *Estação de Testes* tem os objetivos de controlar o processo de aquisição de dados da estação, de analisar os dados através de ajuste de curvas e gráficos estatísticos e de gerar arquivos de *report* com os dados analisados.

O programa, denominado *Chambers 1.0*, foi desenvolvido para plataforma *Windows*. Como características principais, o *Chambers 1.0* deve ter rotinas de acesso à porta-paralela, uma boa interface gráfica e algoritmos rápidos para ajustes de curvas com cerca de 100 parâmetros independentes (§ 4.4).

A interface de desenvolvimento escolhida para a construção do programa *Chambers 1.0* foi o *Delphi 7.0* da *Borland* [33]. O Delphi, além de ser uma das interfaces de desenvolvimento de *software* mais intuitivas para construção de programas com interface gráfica, é baseado em uma poderosa linguagem orientada a objetos, o *Object Pascal*, excelente para desenvolvimento de rotinas de análise de dados.

O código do programa foi estruturado de forma a tirar proveito das características de uma linguagem orientada a objetos. Três classes principais foram desenvolvidas para controlar o processo de aquisição de dados, tornando o acesso ao *hardware* transparente para o restante do programa. Tais classes são listadas abaixo. Para manter a convenção de descrição de classes, seus nomes começam com a letra 'T'.

1. **TF1** - classe que descreve o comportamento do TDC F1. Contém métodos para automatizar a configuração deste conversor, e campos que representam seus registradores internos.

2. **TF1Parallel** - encapsula as rotinas de configuração e aquisição do *Módulo TDC*.

3. **TMultiChannelEPP** - encapsula as rotinas de configuração e aquisição do MCA.

Com exceção de mais uma classe desenvolvida para visualização de gráficos em 3D, a *THeightField*, todo o programa foi construído com componentes nativos da VCL (*Visual Component Library*) do Delphi.As rotinas de ajuste de curvas foram adaptadas do pacote de rotinas matemáticas *Numerical Recipes* para Pascal [48].

A Figura 4-1 mostra a interface gráfica do *Chambers 1.0*. O gráfico principal mostra aquisições em tempo real do *Módulo TDC* ou do *MCA*. Tabelas apresentam os valores dos parâmetros ajustados. Gráficos em 3D representam o mapeamento dos parâmetros sobre os planos da câmara. Não é praticável listar o código-fonte completo do programa em um apêndice, mas o código principal está distribuído nas classes desenvolvidas e apresentadas por completo no Apêndice C.

4.1 A classe TF1

A classe *TF1* contém métodos e campos que serão utilizados pela classe *TF1Parallel* para configuração dos registradores internos do TDC F1. Esta classe faz uso de algumas classes auxiliares como a classe *TStop*, projetada para configuração de alguns parâmetros dos sinais de entrada, e a classe *TPLL*, que é responsável por ajustar a resolução do TDC. As classes auxiliares são descritas no mesmo apêndice que descreve a classe *TF1*. Os campos mais importantes da classe *TF1* são:

1. *Start* - este campo é um objeto da classe auxiliar *TInput* utilizada para ajustar alguns parâmetros do sinal de *start* do TDC como, por exemplo, o nível de *threshold* da entrada inversora.

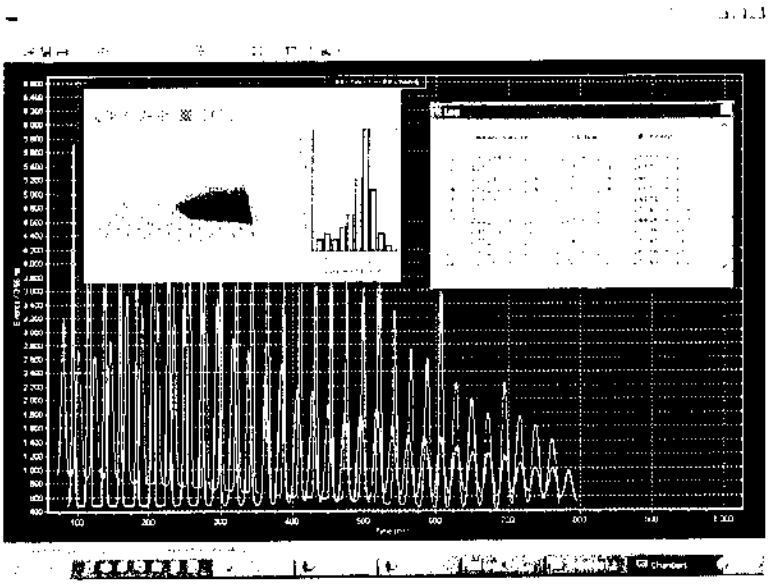

Figura 4-1: Interface gráfica do programa *Chambers 1.0.*

2. *Stop1* a *Stop8* - estes são objetos da classe *TStop*, filha da classe *TInput*. Em *TStop* são acrescentados alguns outros parâmetros de configuração que são usados apenas para os sinais de *stop* do TDC como, por exemplo, o campo *Enable* que diz se o canal está, ou não, habilitado para aquisição.

3. *PLL* - objeto da classe *TPLL*. Contém métodos que controlam o circuito PLL que regula a resolução do TDC.

4.2 A classe TF1Parallel

Esta classe é responsável por controlar todo o processo de aquisição do *Módulo TDC*. Seus campos mais importantes são:

1. **F1** - este é um objeto da classe *TF1*. Os campos deste objeto devem ser configurados de acordo com o modo como o TDC F1 deva funcionar.

2. **Subtraction** - é um campo tipo *boolean*, que diz se a aquisição deve usar, ou não,

as máquinas de estados de subtração, descritas no capítulo anterior.

3. **Stops** - este campo é um ponteiro para uma cópia dos dados histogramados na memória do *Módulo TDC*.

Além dos campos citados acima, essa classe contém vários métodos para controlar a aquisição. São eles:

1. **Start** - configura o TDC com os dados do campo F1 e inicializa o processo de histogramação.

2. **Stop** - pára a histogramação e configura o TDC em modo *power down*.

3. **ClearMemory** - zera os dados da memória do *Módulo TDC*.

4. **ReadMemory** - copia os dados da memória do *Módulo TDC* na memória RAM do PC. Esta classe possui um ponteiro, chamado *Stops*, para este local de memória, de onde os dados podem ser acessados pelo programa.

As instruções de I/O, utilizadas por essas rotinas para acessar o *hardware*, são feitas por um *driver* de dispositivo que detecta automaticamente a versão do Windows e age da forma adequada, fazendo com que o programa funcione para qualquer versão deste sistema operacional, incluindo o Windows XP. Tal *driver* está encapsulado em uma DLL, a *InpOut32.dll*, que pode ser encontrada em [49] com toda documentação necessária.

4.3 A classe TMultiChannelEPP

Assim como a classe *TF1Parallel*, a classe *TMultichannelEPP*, tem o objetivo de acessar o *hardware* e controlar o processo de aquisição. A diferença é que, para esta última classe, o dispositivo alvo é o analisador multicanal. Seus campos principais são:

1. **Window** - um campo do tipo *Integer* que estabelece o tamanho da janela de tempo para a integração do sinal de anodo.

2. **Data** - um ponteiro para uma cópia do histograma do MCA na memória RAM do PC.

Os métodos desta classe, para controle do processo de aquisição do MCA, são iguais aos da classe *TF1Parallel* para aquisição do *Módulo TDC*: **Start**, **Stop**, **ClearMemory** e **ReadMemory**. O código-fonte completo, para esta classe, encontra-se no Apêndice C.

4.4 Rotinas de análise de dados

Espera-se, para o caso de uma câmara iluminada uniformemente, um histograma com 32 picos para cada canal de TDC. Cada pico é uma distribuição normal formada por eventos detectados em um dos 32 *pads* conectado a uma posição específica da linha-de-retardo. Espera-se também uma pequena perda no sinal, à medida em que ele se propaga pela linha. A Figura 2-12 mostra este efeito para um evento coletado nas duas extremidades de uma mesma linha-de-retardo. O pulso coletado na extremidade mais distante do ponto onde o sinal foi injetado, tem menor amplitude e sofre uma maior deformação, pois atravessa um maior número de células. Esta perda resulta em um alargamento dos picos correspondentes a *pads* mais afastados da extremidade da linha onde o sinal é coletado. Porém a integral desta distribuição, que é o parâmetro que fornece o número de eventos por *pad*, é independente deste fenômeno. Para obter este parâmetro com maior precisão, o programa *Chambers 1.0* deve ajustar, a partir do espectro adquirido, uma curva analítica que corresponde a uma soma de 32 gaussianas, sobrepostas a um *background* linear. Com isto, podem-se utilizar os parâmetros ajustados de cada gaussiana para cálculo do valor de sua integral. O capítulo de resultados apresenta alguns destes espectros e suas respectivas curvas, ajustadas pelo programa.

A análise dos dados, através do ajuste de curvas por funções analíticas, foi feita com o auxílio da função de *fit* não linear iterativa, apresentada no *Numerical Recipes* em Pascal, a função *mrqmin*. Esta função requer o uso de funções auxiliares como *mrcof*, *gaussj*, etc, todas apresentadas no *Numerical Recipes*. Estas funções foram estudadas e modificadas

para aceitarem facilidades do *Object Pascal* como *arrays* dinâmicos, tornando-as mais robustas e poderosas.

O Apêndice C mostra todas as funções utilizadas do *Numerical Recipes* devidamente modificadas.Uma função adicional, a *NLFit*, foi criada para facilitar o processo de iteração feito com a função *mrqmin*. A *NLFit* usa, como um dos parâmetros, um ponteiro para função do tipo *TNLFuncs*, que descreve a forma da função base a ser ajustada. No caso do ajuste para os histogramas do TDC, esta função deve ser uma soma de 32 gaussianas e um *background* linear, totalizando 98 parâmetros independentes (3 de cada gaussiana e 2 do *backgeound* linear) a serem ajustados. Estas duas últimas funções também se encontram no Apêndice C.

4.5 Geração de arquivos de report

Depois que os oito espectros de TDC forem adquiridos e as respectivas curvas forem ajustadas, o programa *Chambers 1.0* habilita a opção para geração de um arquivo de *report*. O formato do arquivo gerado é um documento do *Microsoft Word* (*.doc*). Este processo é feito com a ajuda de um componente do *Delphi*, o *TWordDocument*, que auxilia no desenvolvimento de *software* para comunicação com este programa da Microsoft. O *TWordDocument* carrega uma instância do *Word* e gerencia a troca de dados entre este e o programa desenvolvido pelo usuário do componente. Portanto, o *Microsoft Word* deve estar instalado na máquina, para que este componente funcione apropriadamente.

Quando o usuário requer a geração do arquivo de *report*, o programa *Chambers 1.0*, mostra um formulário a ser preenchido com alguns parâmetros importantes. São eles: *Chamber ID*, *Gas mixture*, *High voltage* e *Production center*. A primeira página do *report* apresenta um cabeçalho com os dados do formulário e os oito espectros do TDC com suas respectivas curvas ajustadas. As próximas páginas mostram uma tabela com os resultados dos ajustes para cada plano e gráficos em 3D com a distribuição espacial da eficiência relativa, ao lado de um gráfico estatístico mostrando o desvio padrão desta eficiência. A Figura 4-2 mostra um exemplo da página principal e de uma página com

os resultados para um dos planos da câmara.

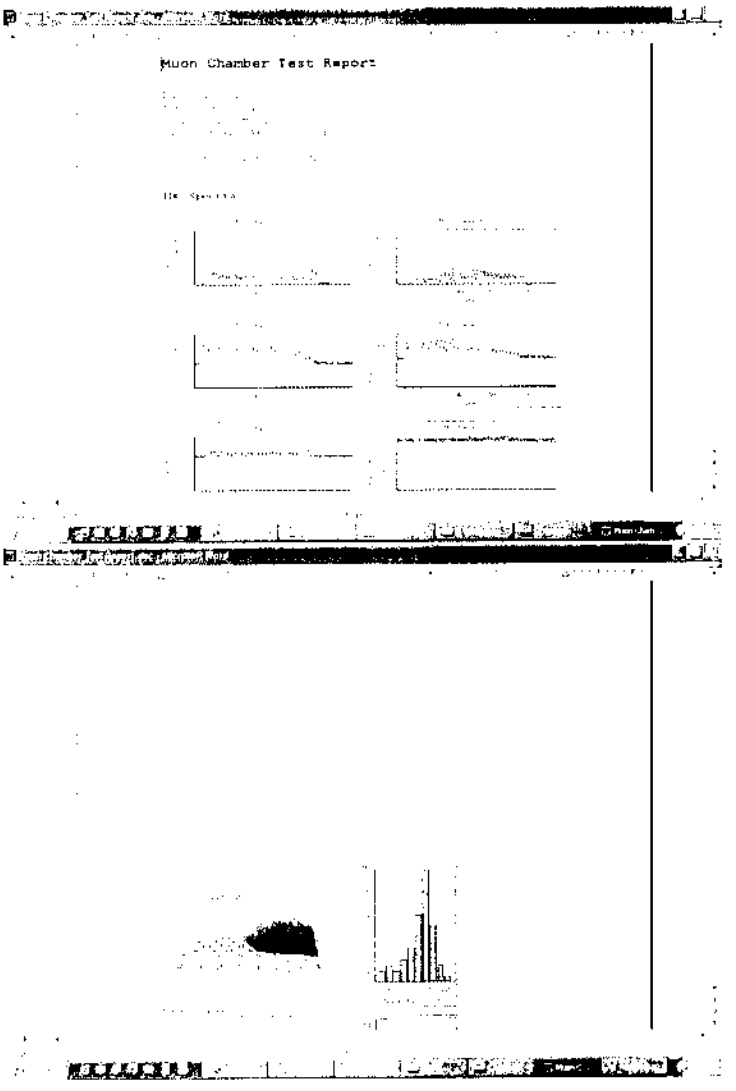

Figura 4-2: Exemplo de um arquivo de *report*, gerado pelo programa *Chambers 1.0*.

Capítulo 5

Resultados

5.1 Desempenho do Módulo TDC

5.1.1 Resolução

A resolução do TDC F1 é definida através de seus registradores de configuração e pode ser ajustada para valores entre cerca de 20% do seu valor nominal de 120 ps. Tal resolução é fixada em 119.61 ps, pelo programa *Chambers 1.0*, segundo a fórmula de cálculo da resolução do F1 [37]. A resolução efetiva, utilizada pela estação de testes, é de $119.61 \times 2^3 \simeq 1$ ns (§ 3.2.1.2). Este valor é suficiente, já que a resolução temporal da câmara é da ordem de 4 ns (§ 2.3.1).

Para medir o valor real da resolução do *Módulo TDC*, foi montado o aparato experimental da Figura 5-1. Neste aparato, o *start* e um dos *stops* do TDC são ligados à saída de um gerador de pulsos (HP8116A). No conector de *start*, o sinal é bifurcado, passa por um cabo de retardo fixo de 16 ns e chega ao conector de *stop*. A Figura 5-2 mostra o espectro resultante para várias frequências do sinal de entrada. Para frequências inferiores a 1 MHz apenas o canal 16 é atingido. A partir desta frequência, até 10 MHz, há um deslocamento gradual do pico para o canal 14. O mesmo comportamento é observado para todos os canais do TDC.

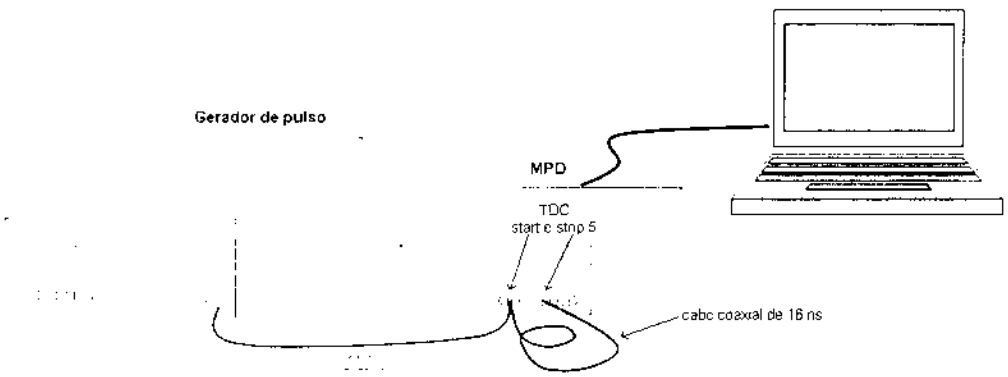

Figura 5-1: *Setup* experimental para medida de resolução do *Módulo TDC*.

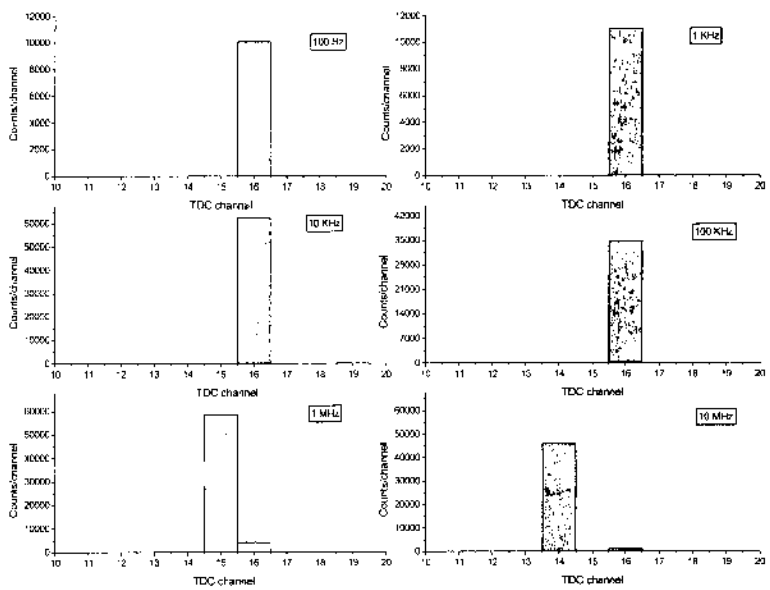

Figura 5-2: Teste de resolução do *Módulo TDC*, feito com um cabo coaxial de 16 ns de retardo para várias frequências do sinal de entrada.

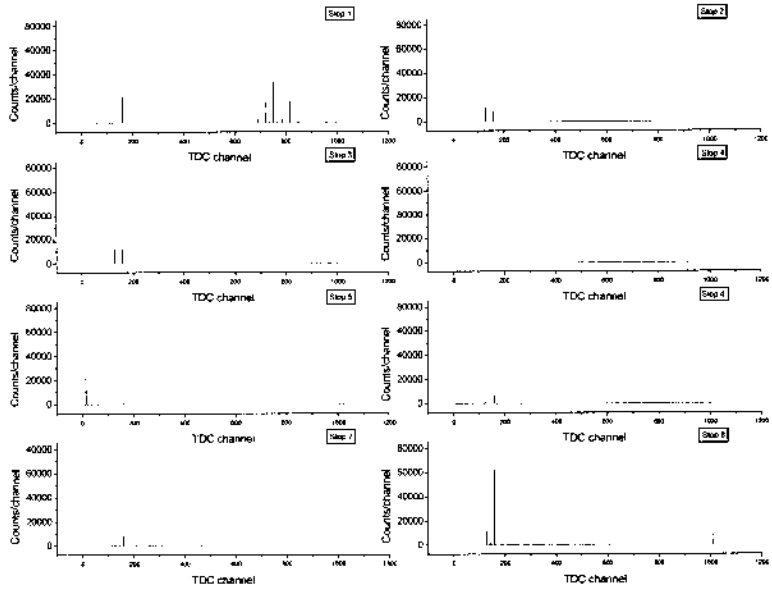

Figura 5-3: Eventos espúrios nos canais do TDC para um sinal de 10 MHz, aplicado ao canal 5.

5.1.2 Taxa de aquisição

Alguns dos pinos do TDC F1 são sinais de *status* que informam se houve perda de dado em algum dos estágios de conversão, devido a altas taxas de eventos na entrada. Um monitoramento destes sinais, feito pelo programa *Chambers1.0*, mostrou que o F1 converte todos os eventos para uma frequência do gerador inferior a 1.91 MHz (*setup* da Figura 5-1), em contradição com o valor de 2 MHz, especificado no manual do F1. A partir desta frequência, observa-se, também, eventos espúrios nos oito canais do TDC (Figura 5-3). O padrão destes espectros sugere um erro de lógica interno, devido a *overflow* nos estágios de conversão, quando a taxa de eventos é maior do que a permitida para o correto funcionamento do TDC.

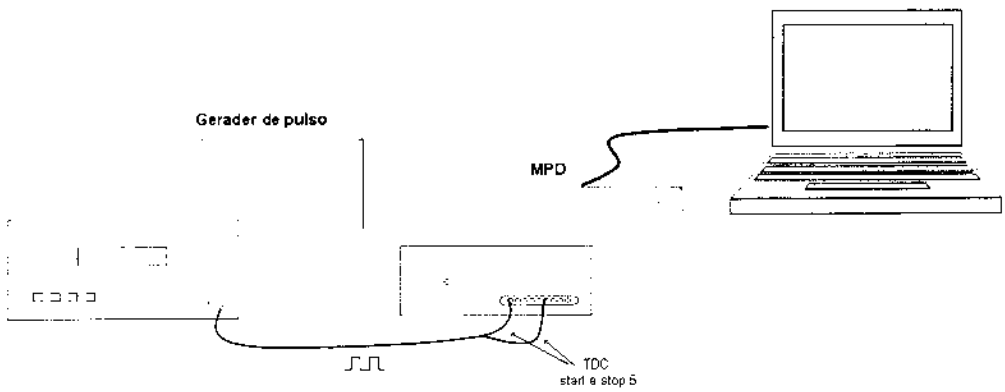

Figura 5-4: *Setup* experimental para medida de linearidade e INL do *Módulo TDC.*

5.1.3 Crosstalk

O *setup* da Figura 5-1 é executado com todos os canais do TDC, habilitados para observação de *crosstalk*. Até a frequência de 1.91 MHz, nenhum evento é observado em outro canal, exceto aquele que recebe o pulso do gerador.

5.1.4 Linearidade

A Figura 5-4 apresenta o arranjo experimental para teste de linearidade do *Módulo TDC*. A saída do gerador de pulsos é bifurcada e conectada ao *start* e a um dos canais do TDC. O *stop* é configurado com *trigger* em transição de descida, para que o tempo medido pelo TDC corresponda à largura do pulso fornecido pelo gerador.

O espectro resultante é obtido variando-se a largura do pulso de 10 ns a 900 ns, em passos de 10 ns (Figura 5-5). Cerca de 55% dos picos têm largura de 2 canais. De acordo com o teste de resolução apresentado acima, esta largura se deve à imprecisão dos pulsos fornecidos pelo gerador. A Figura 5-6 mostra o gráfico de linearidade obtido com este espectro.

Um ajuste linear aplicado ao gráfico da Figura 5-6, construído a partir dos dados obtidos com a medida apresentada na Figura 5-5, é usado para cálculo de linearidade do

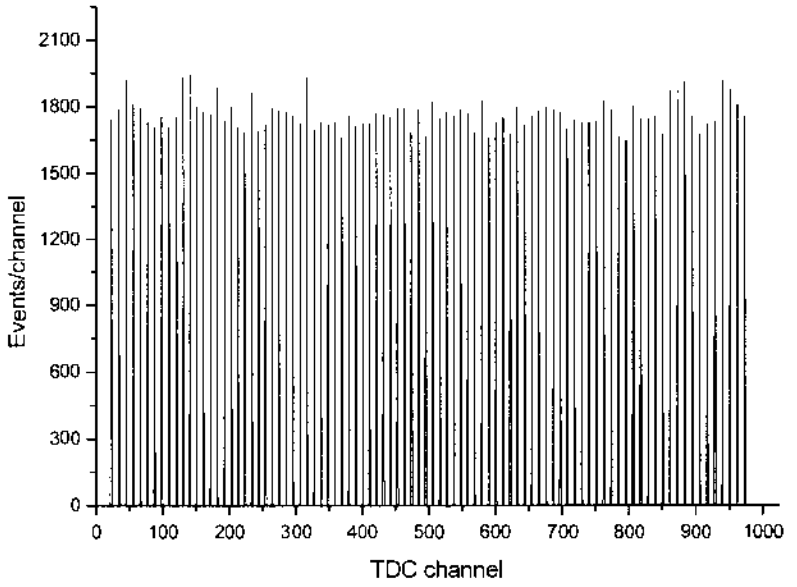

Figura 5-5: Espectro de TDC para medidas de intervalos de tempo de 10 a 900 ns, em passos de 10 ns.

Módulo TDC.

5.1.5 Não linearidade integral

A distribuição normalizada das distâncias entre os pontos medidos e o ajuste linear da Figura 5-6 é apresentada na Figura 5-7. A não linearidade integral (INL - *Integral Non-Linearity*) do sistema é tomada como sendo o maior desvio desta distribuição (0.72 %). Uma parcela deste erro é devida ao gerador de pulsos, portanto, pode-se afirmar que para o *Módulo TDC*

$$INL < 0.72\% \tag{5.1}$$

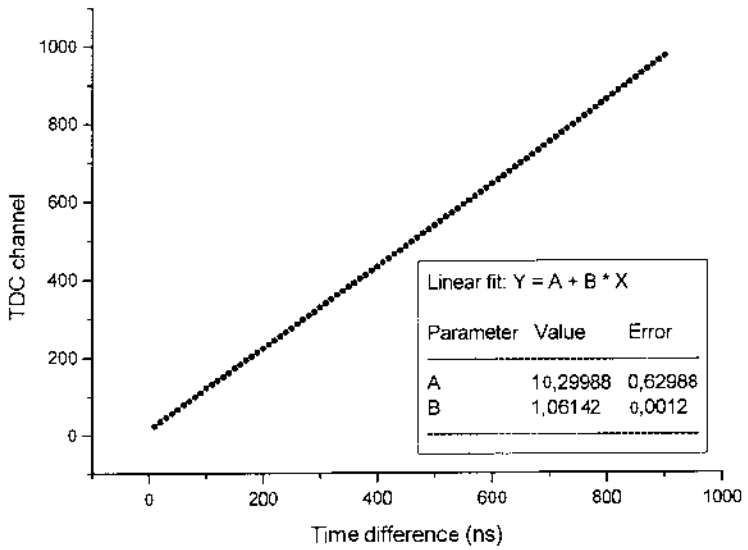

Figura 5-6: Gráfico de linearidade do *Módulo TDC*.

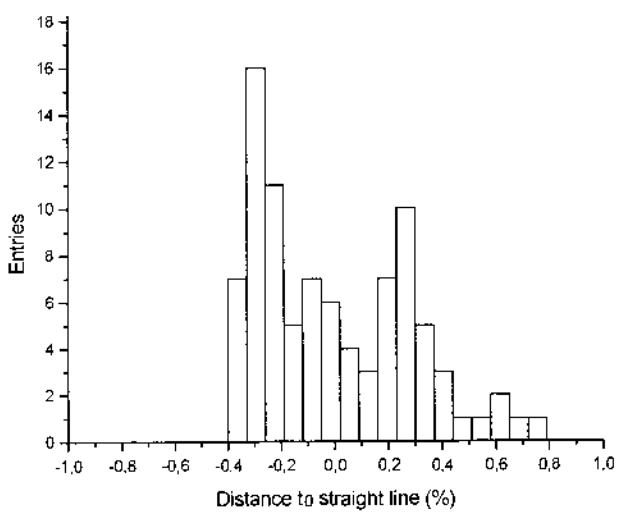

Figura 5-7: Distribuição normalizada das distâncias entre os picos medidos e o ajuste linear para o *Módulo TDC*.

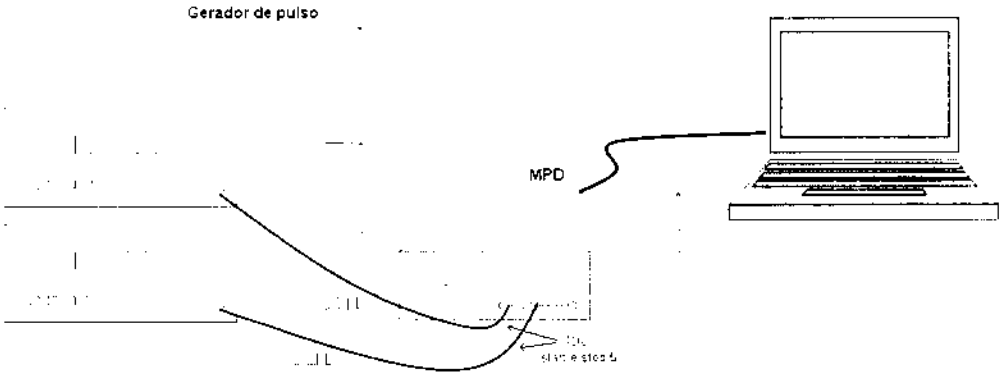

Figura 5-8: Aparato experimental para medida da DNL do *Módulo TDC.*

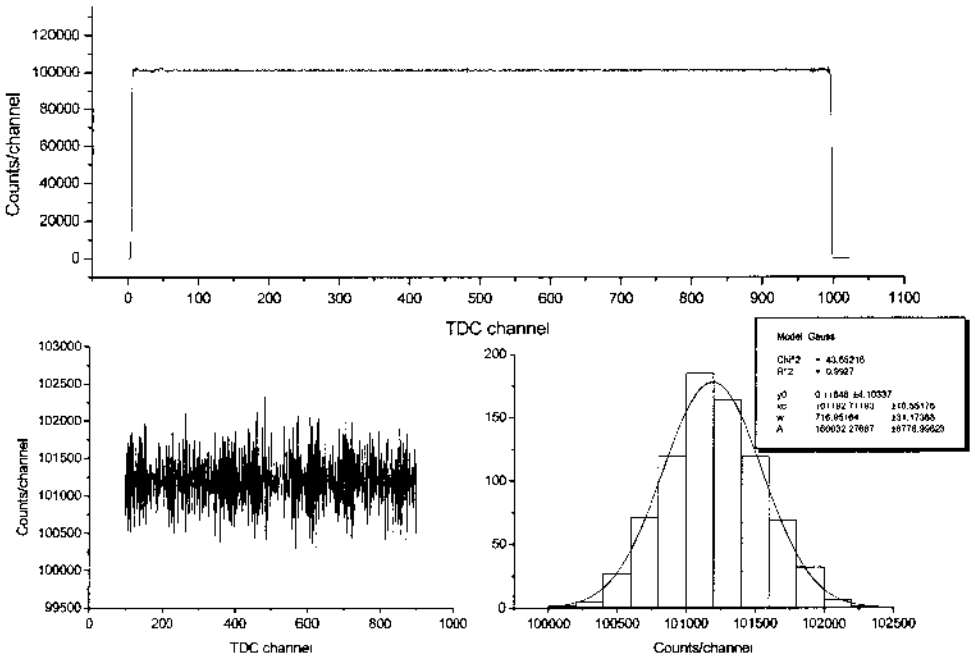

Figura 5-9: Resultado do teste da DNL do Módulo TDC. Gráfico superior - espectro temporal obtido; inferior a esquerda - ampliação da escala vertical; inferior a dircita - distribuição das contagcns.

5.1.6 Não linearidade diferencial

O aparato da Figura 5-8 foi montado para teste da não linearidade diferencial (DNL) do *Módulo TDC*. As entradas de *start* e *stop* foram conectadas separadamente, uma em cada gerador (HP8116A e Agilent 33120A) e configuradas para disparo em transição de subida. Os geradores não estão sincronizados e suas frequências de operação são de 1 MHz e 1.05 MHz, respectivamente. Deste modo, qualquer intervalo de tempo entre 0 e 1 μs pode ser medido com igual probabilidade. O espectro de intervalo de tempo resultante é apresentado na Figura 5-9. Um gráfico estatístico do número de eventos por canal mostra um valor médio de 101192 com um desvio padrão de 358 eventos, ou seja,

$$DNL_1 = \frac{358}{101192} \times 100 = 0.35\% \tag{5.2}$$

A DNL tomada a partir do desvio máximo de 1106 eventos, presente no canal 482, é

$$DNL_2 = \frac{1106}{10192} \times 100 = 1.09\% \tag{5.3}$$

5.1.7 Intervalo de tempo mínimo

O resultado da Figura 5-9 mostra também o intervalo de tempo mínimo que o F1 é capaz de medir. Um *zoom* feito nos primeiros canais do espectro (Figura 5-10) indica um intervalo de tempo mínimo da ordem de 5 ns, que é compatível com as especificações do F1.

5.2 Desempenho do MCA

5.2.1 Calibração em carga e linearidade

Para que o MCA forneça a informação em unidade de carga, ele deve ser submetido a um processo de calibração, utilizando um pulso de amplitude fixa.Para isto, é necessária uma alteração no código da FPGA de modo a desprezar as primeiras amostras após o disparo

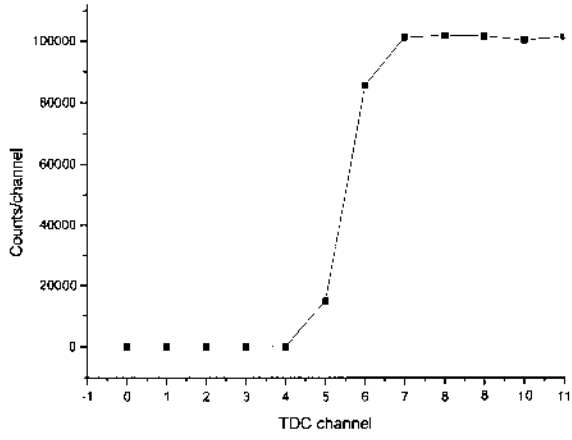

Figura 5-10: Ampliação da escala horizontal do espectro temporal da Figura 5-9, mostrando que o intervalo de tempo mínimo que o F1 é capaz de medir é de ≈ 5 ns.

do discriminador. Esta alteração tem por objetivo eliminar amostras que eventualmente atinjam a região de transição do pulso, o que resultaria em uma informação inválida, uma vez que esta região contém componentes de frequências superiores à metade da taxa de amostragem do ADC.

O *setup* da Figura 5-11 foi montado para realizar a calibração em carga do MCA. Neste aparato, utilizou-se uma janela de 500 ns para integração do sinal amostrado pelo ADC a 40 MHz (20 amostras). A largura da janela é configurada pelo programa *Chambers 1.0*. Desta forma, a Equação 2.16 pode ser escrita como

$$q = \frac{500}{R} \times V \tag{5.4}$$

para uma terminação $R = 50\Omega$ e V em milivolts tem-se

$$q = 10 \times V \quad (pC) \tag{5.5}$$

A medida mostrada na Figura 5-12 foi realizada, variando-se a amplitude do pulso de

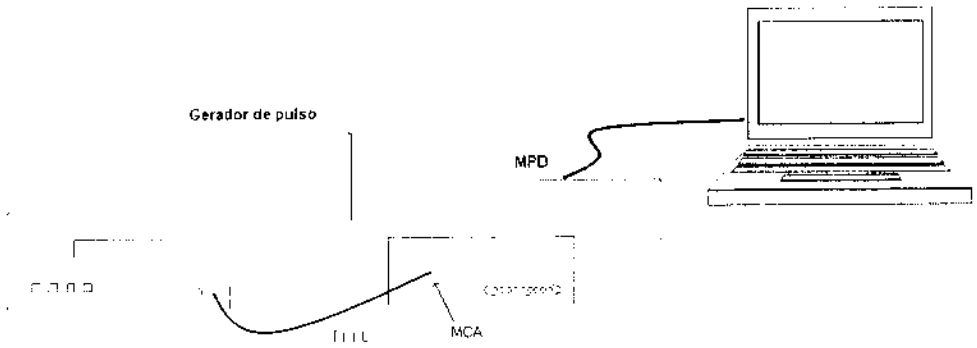

Figura 5-11: Aparato experimental para calibração do MCA.

70 mV a 550 mV em passos de 20 mV, o que equivale a variar a carga de 700 pC a 5500 pC em passos de 200 pC (Equação 5.5). A frequência do gerador se mantém fixa em 1 kHz. Cada pico resultante é uma distribuição normal com largura máxima (2σ) de 3.5 canais. A Figura 5-13 mostra o resultado da calibração em carga do MCA.

5.2.2 Resolução

O resultado de calibração, dado pelo ajuste linear da Figura 5-13, mostra que cada canal do MCA corresponde a uma carga de

$$\Delta q \simeq \frac{1}{0.15} = 6.67\ pC \tag{5.6}$$

Porém, como cada pico tem uma largura máxima de 3.5 canais (Figura 5-12), tem-se que a resolução real do sistema MCA + gerador é de

$$\Delta q_{\text{real}} \simeq 6.67 \times 3.5 = 23.35\ pC$$

Espera-se, portanto, para o MCA, uma resolução inferior a 23.35 pC.

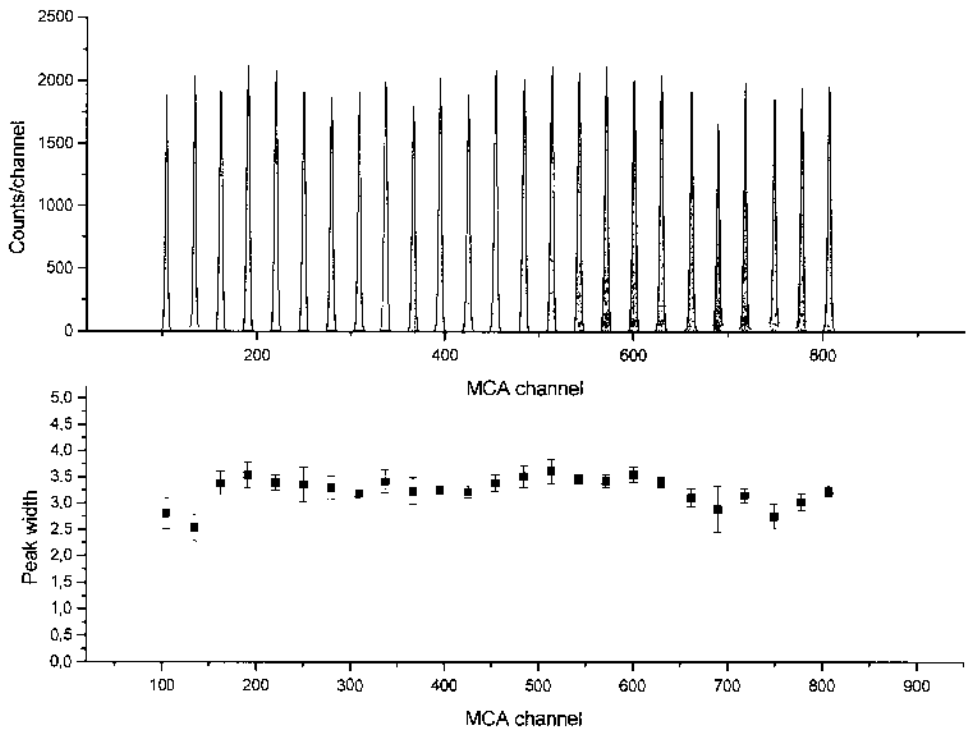

Figura 5-12: Espectro do MCA para medidas de carga de 700 a 5500 pC, em passos de 200 pC. O gráfico inferior mostra o valor da largura (2σ) de cada gaussiana ajustada, com seus respectivos erros obtidos no processo de *fit*.

5.2.3 Não linearidade Integral

A distribuição normalizada das distâncias (Figura 5-14) entre os valores dos picos medidos e o ajuste linear da Figura 5-13, indica uma INL de 0.22 % para o sistema (gerador + MPD). Portanto, para o MCA tem-se

$$INL < 0.22\% \tag{5.7}$$

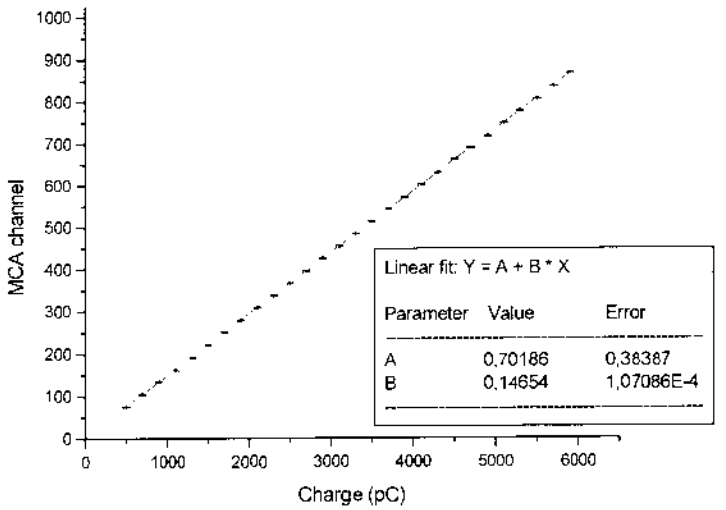

Figura 5-13: Resultado da calibração em carga do MCA.

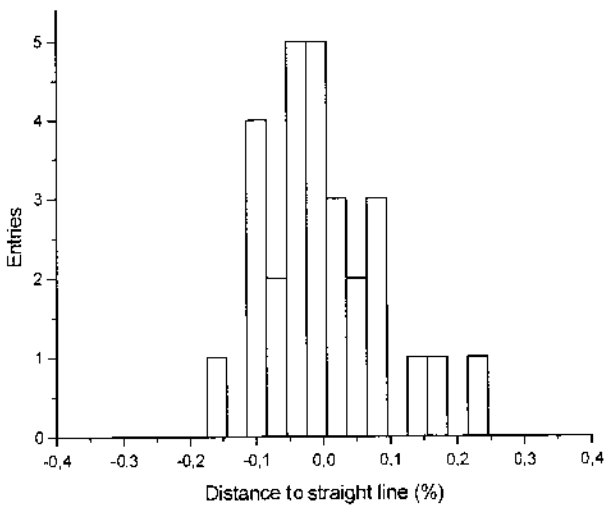

Figura 5-14: Distribuição normalizada das distâncias entre os valores dos picos medidos e o ajuste linear do MCA.

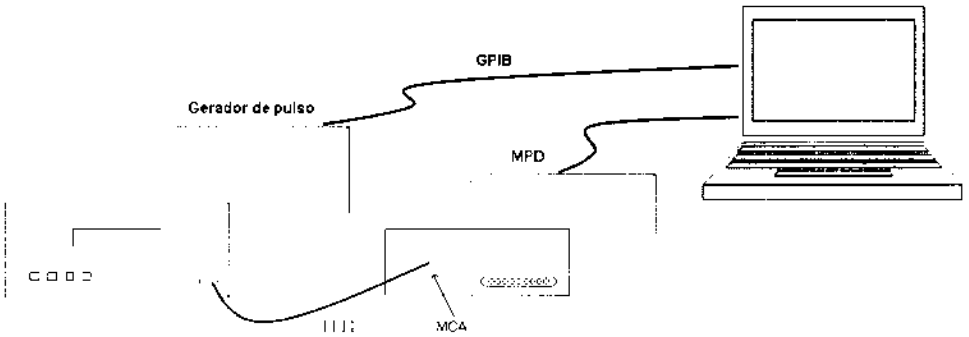

Figura 5-15: Aparato experimental para medida da DNL do MCA.

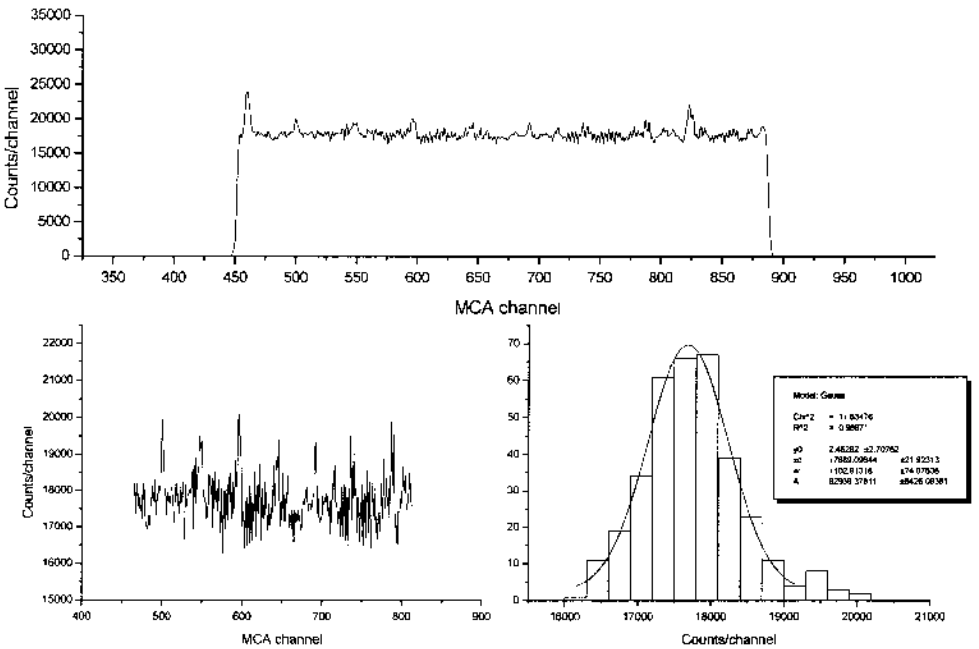

Figura 5-16: Resultado do teste da DNL do MCA + gerador de pulsos.

5.2.4 Não linearidade diferencial

Para medir a DNL do MCA, a amplitude do pulso quadrado, fornecido pelo gerador, é controlado por um programa em *LabView* via barramento GPIB (Figura 5-15). O programa gera números inteiros aleatórios, entre 300 e 600, que são utilizados como valores para a amplitude do pulso em mV. A Figura 5-16 apresenta o resultado. Um ajuste gaussiano indica um valor médio de 17689 com um desvio padrão de 551 eventos, ou seja,

$$DNL_s = \frac{551}{17689} \times 100 = 3.11\%$$

para o sistema gerador + MCA. Grande parte desta não linearidade é devido ao aparato experimental utilizado para se fazer esta medida. Portanto, para o MCA tem-se

$$DNL < 3.11\%$$

5.2.5 Taxa de aquisição

A taxa máxima de aquisição permitida pelo MCA desenvolvido depende somente da largura da janela utilizada para integração do sinal de entrada. No caso típico de uma janela de 500 ns, esta taxa é de dois milhões de aquisições por segundo (2 MHz).

5.2.6 Comparação com um MCA comercial

Para comprovar a qualidade do MCA desenvolvido para uso em medidas de ganho, foi feita uma comparação entre o sistema desenvolvido e um sistema comercial. Os dois MCAs foram submetidos a teste de medida de ganho em um detector proporcional. Utilizou-se, neste teste, o detector monofilar da Figura 5-17, por apresentar uma melhor resolução em energia em comparação com a câmara de múons do LHCb. Optou-se também pela utilização de uma fonte de ^{55}Fe por apresentar dois picos de energias bem definidas (2.7 e 5.9 keV).

O teste consiste em observar a variação da energia depositada pela partícula incidente

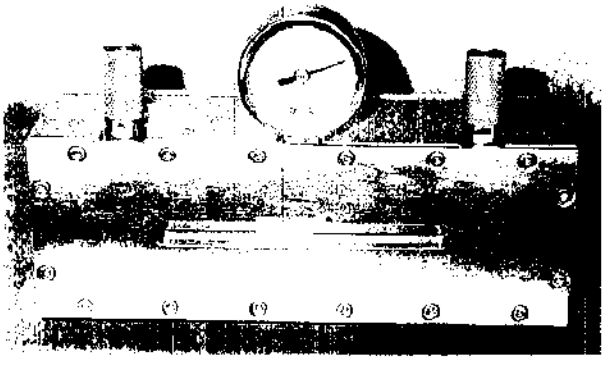

Figura 5-17: Detetor linear proporcional, utilizado para teste comparativo entre o MCA desenvolvido e um MCA comercial

com a variação da alta tensão aplicada ao detector. Pode-se mostrar que esta relação é exponencial para detectores operando na região proporcional [31]. É conhecido que o MCA comercial é capaz de fornecer esta relação corretamente. O Teste (Figura 5-18) foi feito variando-se a alta tensão de 2000 a 2180 V em passos de 20 V. Os 10 espectros resultantes são apresentados na Figura 5-19. À medida em que a alta tensão aumenta, os picos deslocam-se para canais à direita do espectro. Um ajuste gaussiano foi aplicado aos picos de 2.7 e 5.9 keV para facilitar a localização do seu ponto central. O gráfico do valor médio dos picos em função da alta tensão aplicada mostra o comportamento exponencial esperado (Figura 5-20). Nota-se, por comparação entre os parâmetros b dos *fits* exponenciais, que a medida de ganho é compatível nos dois sistemas.

5.3 Resultados com a Estação de Testes

5.3.1 Desempenho do sistema para medida de uniformidade de eficiência

O sistema utilizado para teste de eficiência relativa das câmaras de múons é composto de linha-de-retardo, pré-amplificador, amplificador+discriminador e do *Módulo TDC*. O aparato experimental da Figura 5-21 foi montado para medir o desempenho deste sistema.

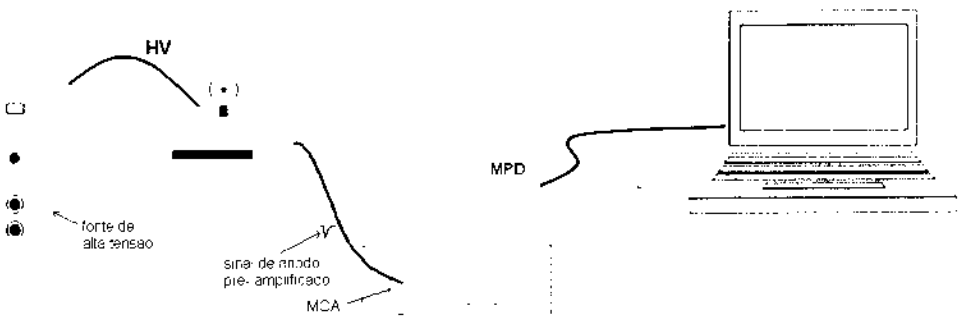

Figura 5-18: *Setup* utilizado para comparação de medida de ganho entre o MCA desenvolvido e um MCA comercial.

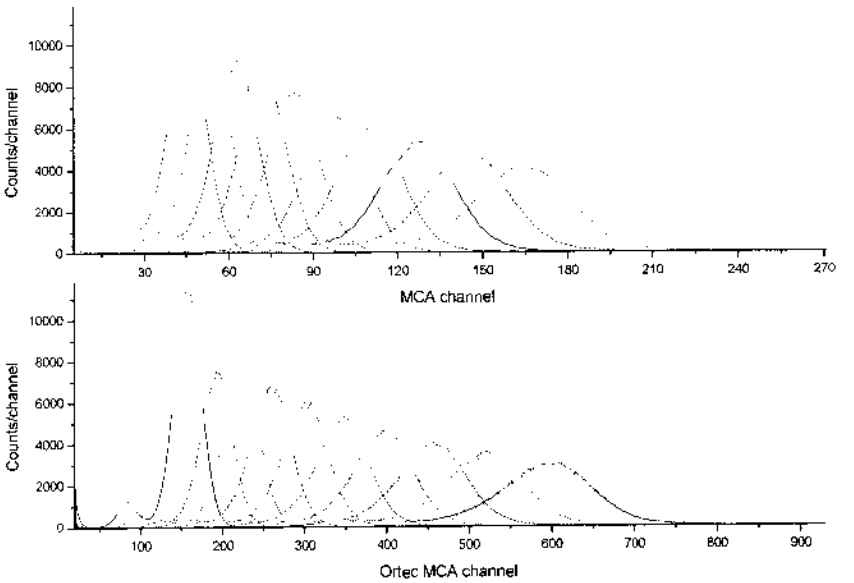

Figura 5-19: Medida de resolução em energia feita com um detetor proporcional e fonte de ^{55}Fe. Cada gráfico mostra espectros para 10 valores de alta tensão aplicada. O gráfico superior e inferior foram feitos com o MCA desenvolvido e um MCA comercial, respectivamente

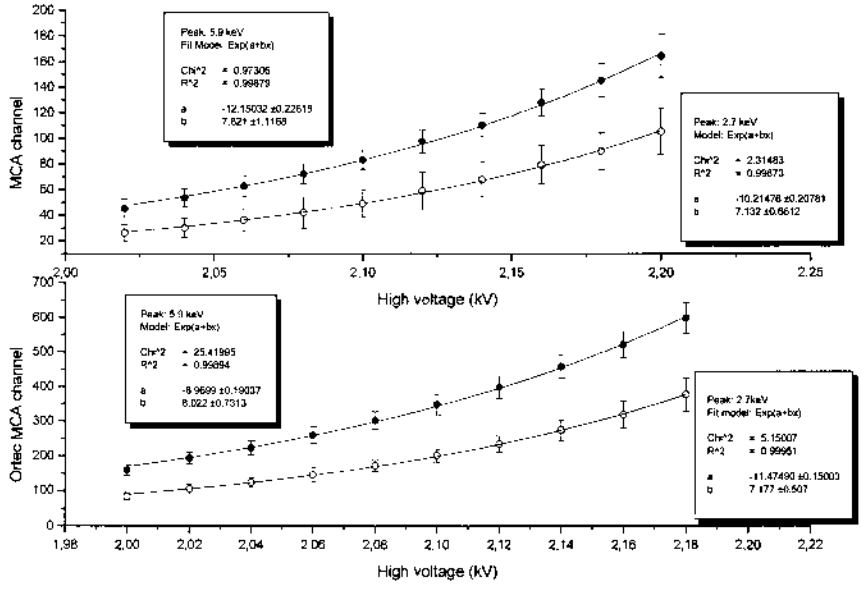

Figura 5-20: Gráfico com as posições dos picos de ^{55}Fe pela alta tensão aplicada ao detector, para o MCA desenvolvido (acima) e um MCA comercial (abaixo).

Como as capacitâncias dos *pads* de catodo atuam na função de transferência das células de retardo, a linha deve estar devidamente conectada à câmara para uma correta obtenção dos resultados.

Um gerador de pulsos é utilizado para injetar um sinal em pontos da linha-de-retardo, correspondentes aos 32 *pads* de catodo. Injetando-se pulsos de 100 ns e amplitude de 10 mV, obtém-se o espectro da Figura 5-22. Os picos resultantes estão espaçados de 24 canais ($\approx$ 24 ns) e têm cerca de 1.4 canais de largura.

5.3.2 Medidas de eficiência relativa

Esta seção apresenta medidas de eficiência relativa, utilizando uma câmara protótipo da região M3R1 do *Sistema de Múons* do LHCb.A Figura 5-23 mostra o espectro temporal de uma aquisição com raios cósmicos, para uma das linhas-de-retardo acoplada à câmara.

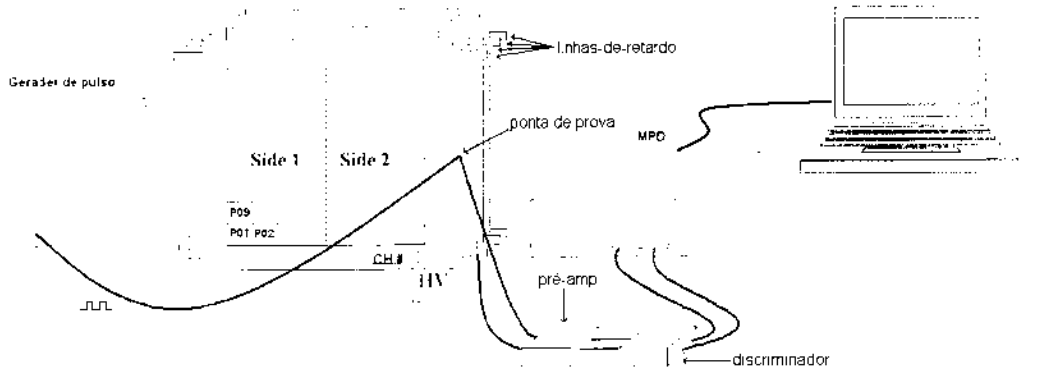

Figura 5-21: Aparato experimental, ulilizado para teste de desempenho do sistema de medida de eficiência relativa

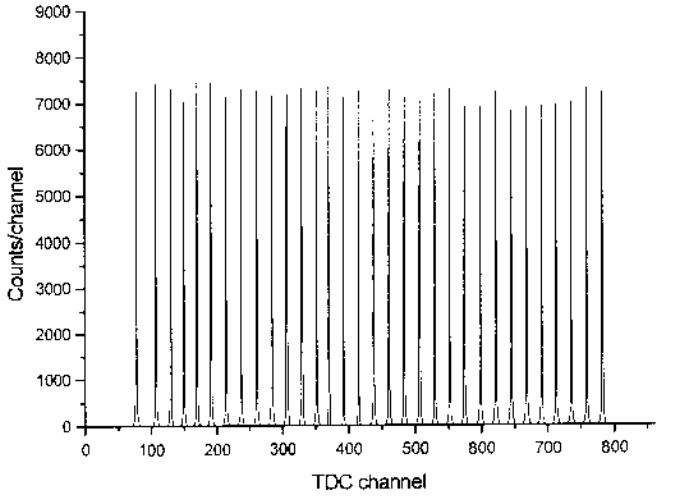

Figura 5-22: Espectro obtido no teste de desempenho do sistema de medida de eficiência relativa, injetando-se pulsos em pontos da linha-de-retardo, correspondentes à conexão dos 32 *pads* de catodo.

Um ajuste de 32 gaussianas e um *background* linear (linha em cor preta), feito pelo programa *Chambers 1.0*, mostra, como esperado, um alargamento das gaussianas para picos mais distantes da extremidade da linha onde o sinal é coletado (§ 4.4). A tabela da Figura 5-24, gerada por este programa, mostra os parâmetros individuais de cada pico e do *background* linear. A distribuição do número de eventos por pico (Figura 5-25) apresenta uma variação na eficiência de

$$Ef = \frac{49.64}{762.17} \times 100 = 6.51\%$$

utilizando o desvio padrão. Porém, 37 % dos *pads* apresentam número de eventos fora desta região e observa-se um desvio máximo de 39% no *pad* 26.

Um outro papel fundamental deste sistema é a identificação de *pads* defeituosos. A Figura 5-26 mostra, como exemplo, um *pad* altamente ruidoso, em uma outra região da mesma câmara protótipo. Esta alta taxa de contagem pode ser devida a impurezas depositadas sobre o *pad* ou sobre os fios nesta região, ou ainda, a descargas causadas por defeitos geométricos introduzidos no processo de fabricação da câmara.

5.3.3 Medida de ganho

O primeiro teste de ganho, feito pela estação, é identificar se a câmara opera corretamente na região proporcional. O procedimento é o mesmo apresentado em (§ 5.2.6), e espera-se, portanto, um comportamento exponencial.O teste é feito, iluminando-se a câmara uniformemente com uma fonte de ^{241}Am. A Figura 5-27 mostra os espectros resultantes com um ajuste gaussiano aplicado ao pico do espectro de amerício, para cada tensão aplicada. O valor médio de cada distribuição normal é utilizado para gerar o gráfico da Figura 5-28, cujo *fit* exponencial, indica que a câmara opera adequadamente na região proporcional.

Se um único *pad* é iluminado de cada vez, e a alta tensão é fixada em um valor determinado, obtém-se um espectro de energia para cada *pad* de catodo. A distribuição da posição dos picos de energia da fonte utilizada, fornece a uniformidade de ganho da

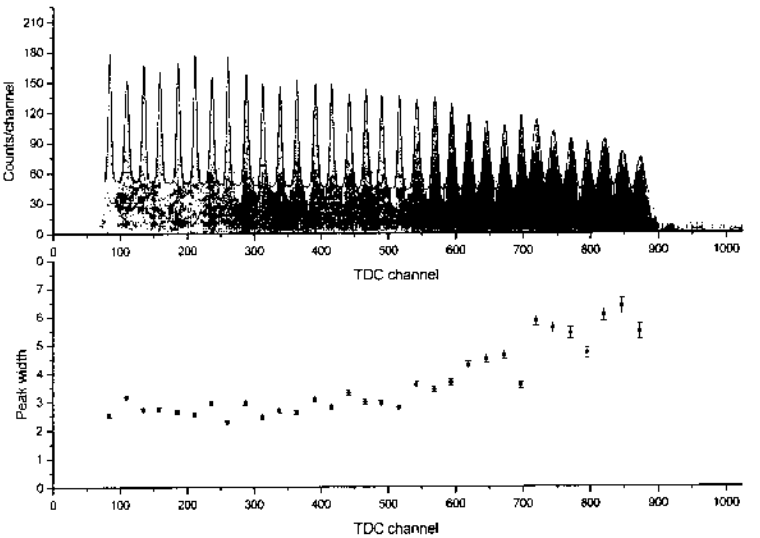

Figura 5-23: Espectro de TDC resultante para uma câmara protótipo M3R1, utilizando-se raios cósmicos. O gráfico inferior mostra o aumento esperado na largura dos picos resultantes.

câmara. A Figura 5-29 mostra o resultado de uma medida de uniformidade de ganho feita em um dos planos da câmara protótipo. A aquisição foi feita com uma fonte de ^{214}Am e cada espectro foi adquirido durante 40 segundos, de modo que o procedimento total levou cerca de uma hora. Note-se que a variação da posição do pico está dentro de 14 canais, correspondendo a uma variação de ganho inferior a 30%. A Figura 5-30 mostra uma visualização tridimensional da medida de uniformidade de ganho.

Channel 1 Fit: chi2/ndf = 1.59

	mean value	sigma	# events
1	83.91 ~ 0.10	2.52 ~ 0.10	795 ~ 41
2	109.51 ~ 0.18	3.15 ~ 0.18	803 ~ 44
3	134.64 ~ 0.15	2.71 ~ 0.14	899 ~ 41
4	158.27 ~ 0.16	2.73 ~ 0.16	749 ~ 40
5	185.24 ~ 0.15	2.63 ~ 0.14	784 ~ 40
6	210.33 ~ 0.14	2.56 ~ 0.13	839 ~ 40
7	235.99 ~ 0.17	2.95 ~ 0.16	797 ~ 41
8	259.84 ~ 0.13	2.29 ~ 0.13	735 ~ 37
9	287.41 ~ 0.17	2.95 ~ 0.16	819 ~ 41
10	311.83 ~ 0.16	2.86 ~ 0.15	638 ~ 36
11	337.73 ~ 0.17	2.68 ~ 0.16	669 ~ 38
12	362.74 ~ 0.16	2.61 ~ 0.15	691 ~ 38
13	390.58 ~ 0.18	3.07 ~ 0.17	791 ~ 41
14	415.03 ~ 0.17	2.77 ~ 0.16	712 ~ 38
15	441.13 ~ 0.20	3.29 ~ 0.19	769 ~ 41
16	465.95 ~ 0.19	2.98 ~ 0.17	732 ~ 39
17	489.33 ~ 0.18	2.94 ~ 0.17	699 ~ 39
18	515.66 ~ 0.18	2.76 ~ 0.17	644 ~ 38
19	541.79 ~ 0.21	3.59 ~ 0.20	880 ~ 43
20	568.37 ~ 0.20	3.43 ~ 0.19	801 ~ 43
21	593.02 ~ 0.22	3.66 ~ 0.21	702 ~ 40
22	618.43 ~ 0.26	4.28 ~ 0.25	812 ~ 07
23	646.70 ~ 0.26	4.58 ~ 0.28	784 ~ 49
24	671.63 ~ 0.39	4.63 ~ 0.39	765 ~ 49
25	696.39 ~ 0.23	3.56 ~ 0.23	690 ~ 43
26	718.51 ~ 0.31	5.83 ~ 0.33	1058 ~ 58
27	743.48 ~ 0.35	5.59 ~ 0.37	855 ~ 56
28	769.47 ~ 0.37	5.40 ~ 0.39	744 ~ 56
29	794.44 ~ 6.36	4.79 ~ 9.39	611 ~ 59
30	820.91 ~ 0.48	6.05 ~ 0.43	935 ~ 61
31	846.35 ~ 0.51	6.35 ~ 0.55	682 ~ 63
32	972.78 ~ 0.50	5.46 ~ 0.55	524 ~ 58

	linear coeff.	angular coeff.
background	54.49 ~ 1.11	-0.0282 ~ 0.0029

Figura 5-24: Tabela com os resultados do ajuste de 32 gaussianas sobrepostas a um *background* linear, gerada pelo programa *Chambers 1.0*.

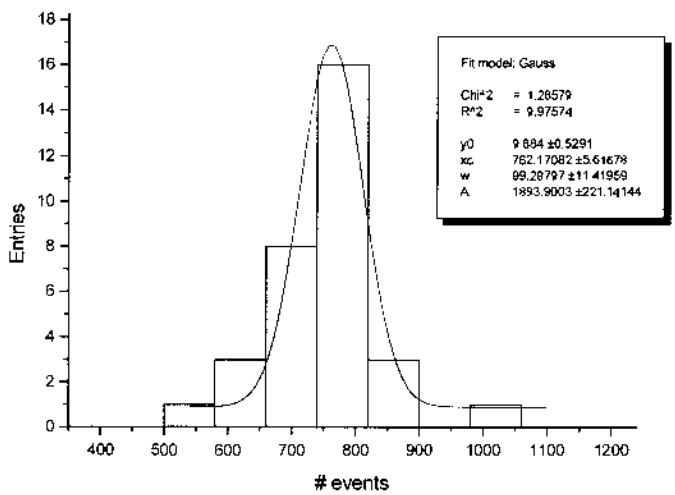

Figura 5-25: Ajuste gaussiano feito para a distribuição do número de eventos por pico.

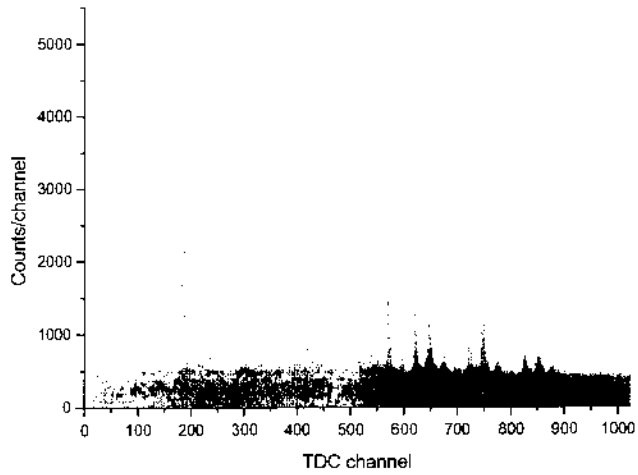

Figura 5-26: Espectro de TDC, mostrando defeito em um dos *pads*, para uma câmara protótipo.

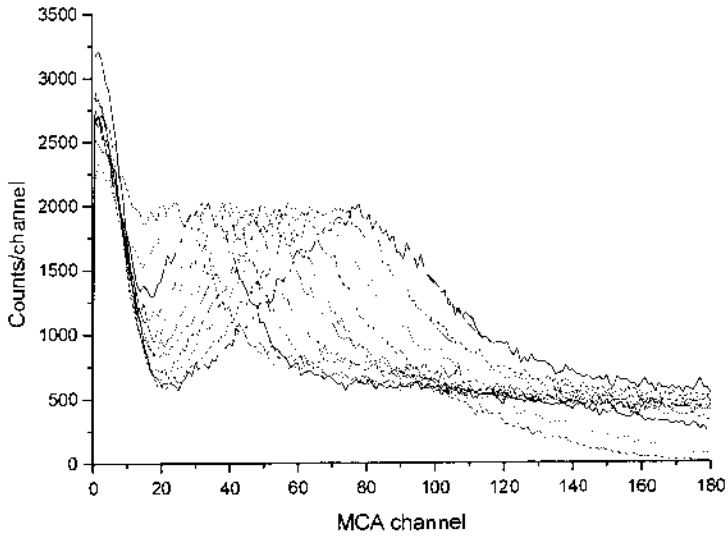

Figura 5-27: Medida de ganho feito com uma fonte ^{241}Am, para uma câmara protótipo. Cada espectro corresponde a uma alta tensão aplicada à câmara.

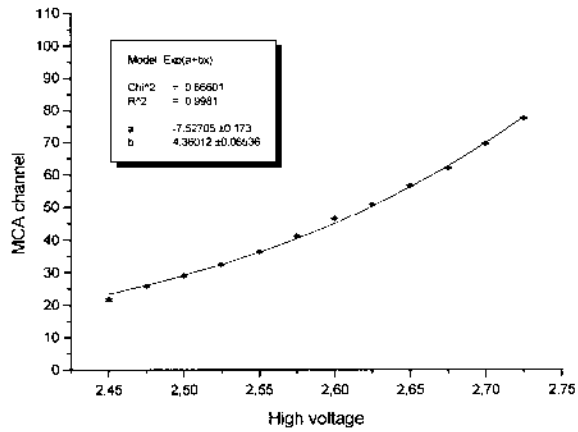

Figura 5-28: Gráfico com as posições do pico de ^{241}Am pela alta tensão aplicada ao detector, para o MCA desenvolvido. O erro é dado pelo próprio algoritmo de *fit*, para o valor médio de cada gaussiana.

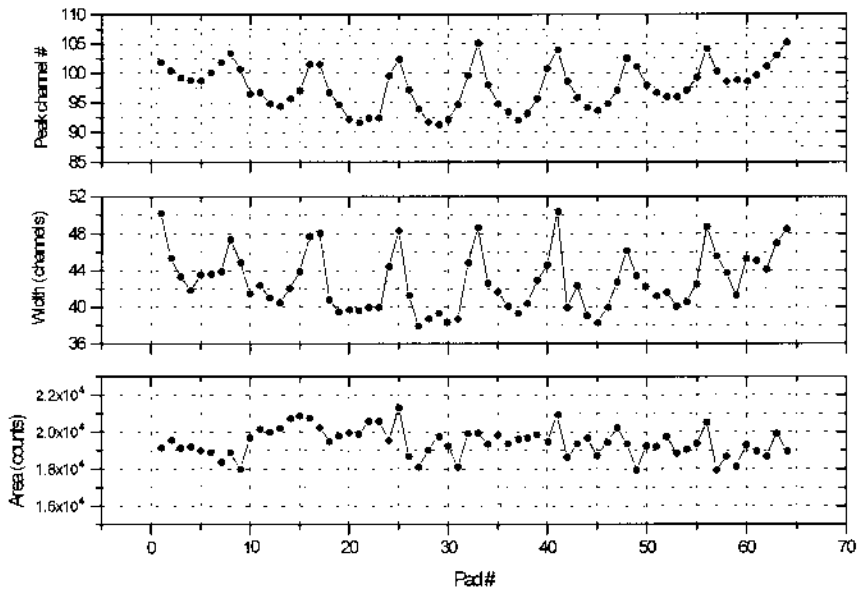

Figura 5-29: Medida de uniformidade de ganho, tomada no *gap* 1 da câmara protótipo.

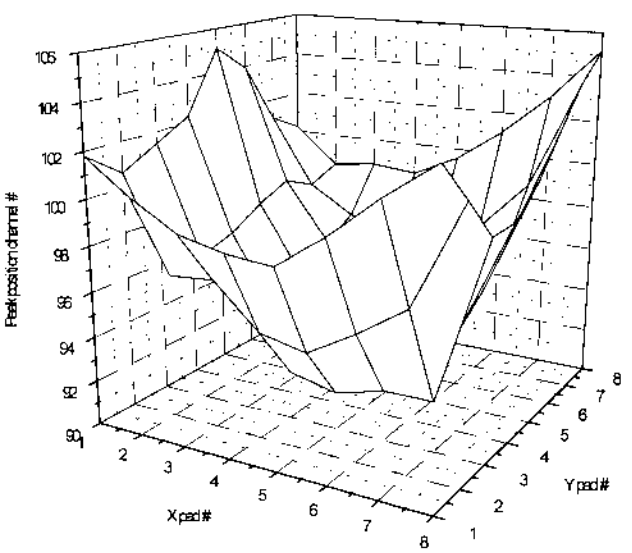

Figura 5-30: Visualização tridimensional da uniformidade de ganho, medida em um dos planos de uma câmara de múons do LHCb.

Conclusão

Os testes de desempenho do *Módulo TDC* e do *MCA*, apresentados no capítulo de resultados, mostram que o MPD atende as especificações de projeto da *Estação de Testes*. Conclui-se, portanto, que o trabalho proposto nesta tese, ou seja, o processamento e análise dos dados da estação, foi realizado com sucesso.

A resolução de 1 ns do *Módulo TDC* mostra que este sistema está perfeitamente qualificado para o processo de identificação de *pads* de catodo, onde uma incerteza de 4 ns é imposta pela resolução temporal da câmara. Além disso, os cuidados na confecção do *layout*, como a utilização de planos internos para alimentação e terra, possibilitaram a eliminação de *crosstalk* entre canais do TDC, para toda a faixa de operação deste circuito (1.91 MHz). Estes cuidados também resultaram em uma ótima linearidade do sistema, com DNL e INL inferiores a 1%. A constatação de que o mínimo intervalo de tempo que o F1 é capaz de medir é de 5 ns, não influi nos espectros temporais obtidos com a linha-de-retardo. Isto ocorre graças à presença de células de guarda em suas extremidades, fazendo com que nenhum evento válido seja observado em um intervalo de tempo inferior a 100 ns.

O *MCA* apresenta uma resolução real, em carga, inferior a 23 pC, que é pequena o suficiente para medidas de ganho com a câmara de múons. Este fato foi comprovado, de forma bastante rigorosa, comparando-se os resultados de medida de ganho com um MCA comercial, em um detector com resolução em energia superior à da câmara de múons do LHCb. A reduzida INL de 0.2% do sistema se deve ao processo de integração do sinal de entrada, onde o acúmulo de dados convertidos tende a compensar a não linearidade do ADC. A DNL próxima de 3.0% é compatível com a maioria dos MCAs comerciais.

A título de perspectiva, uma pequena modificação pode ser feita ao MPD, com o objetivo de sincronizar os eventos do MCA com os do *Módulo TDC*. Desta forma, o TDC seria usado para identificar a posição do evento, enquanto o MCA usaria esta informação para histogramar o evento no espectro de ganho do *pad* correspondente. Assim sendo, a medida de uniformidade de ganho pode ser obtida iluminando-se todos os *pads* da câmara uniformemente. Esta alteração ja está sendo realizada e o sistema modificado deve ser utilizado para as novas câmaras que estão sendo montadas atualmente no CERN.

Apêndice A

Esquemáticos

A.1 Pré-amplificador

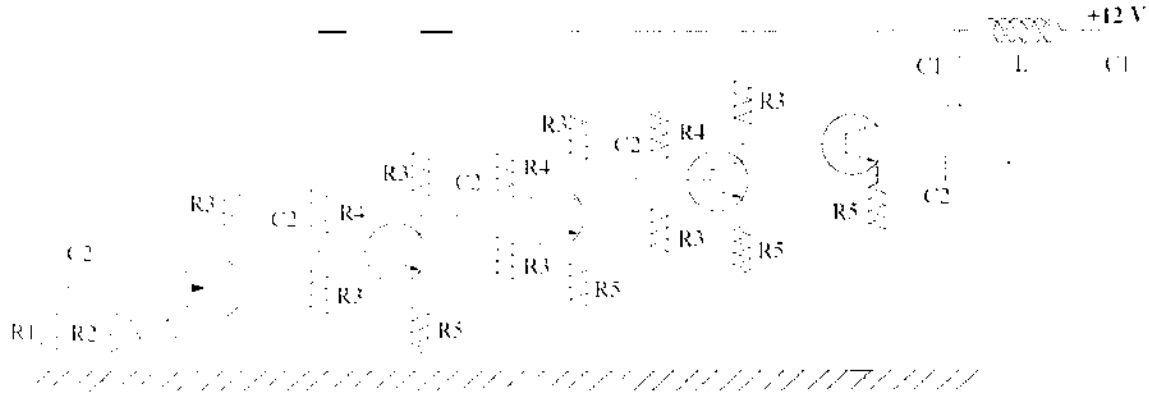

Figura A-1: Esquemático do pré-amplificador utilizado na *Estação de Testes*.

A.2 Amplificador-discriminador

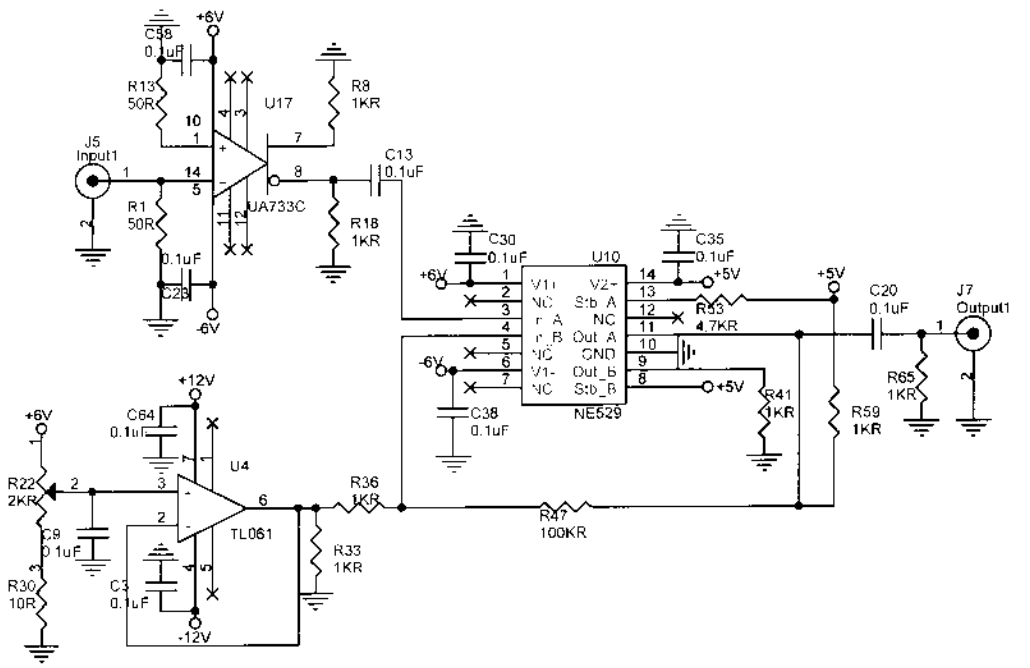

Figura A-2: Esquemático de um canal de amplificação-discriminação. O circuito completo, projetado para a *Estação de Testes*, possui cinco canais.

A.3 Conversor TDC

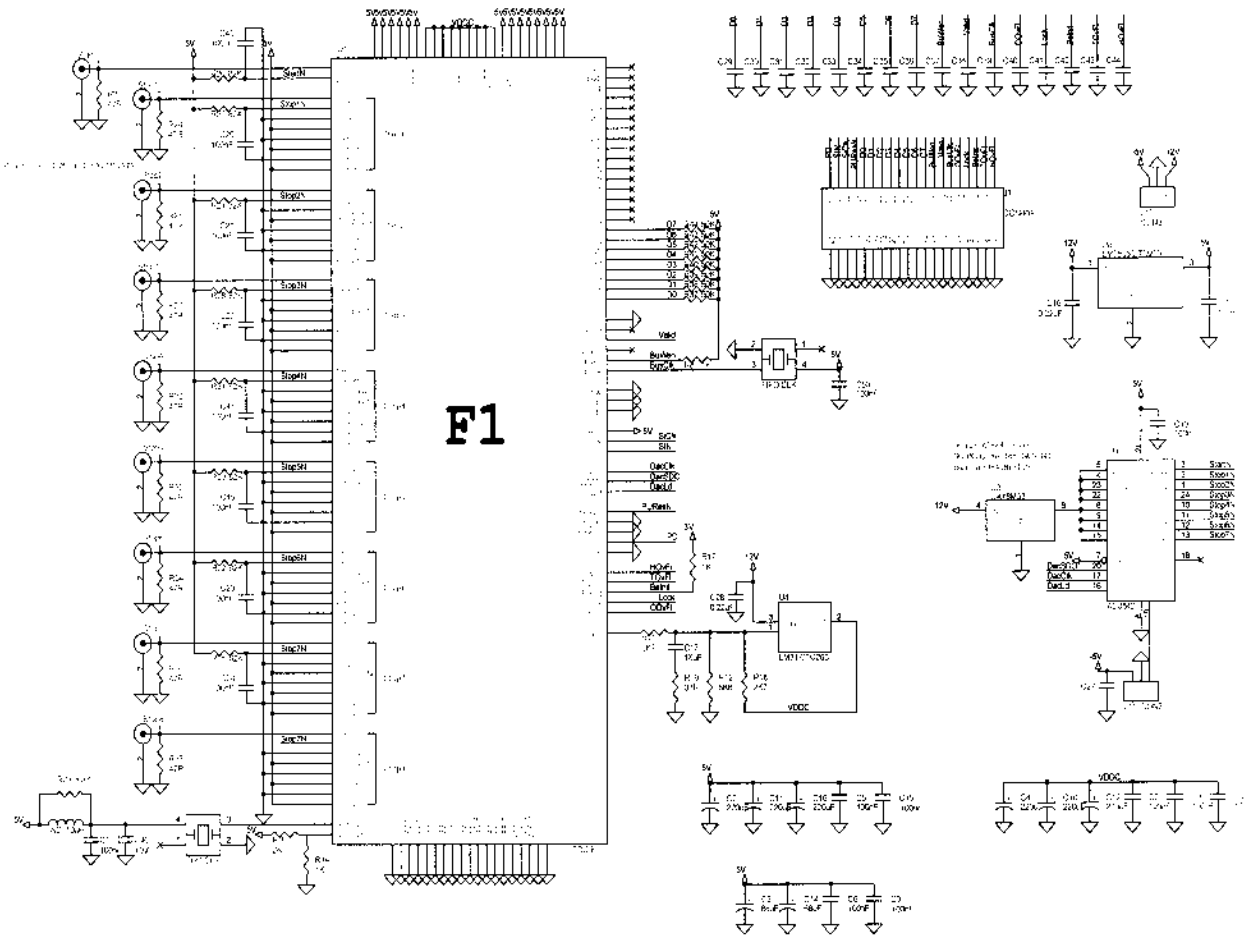

Figura A-3: Esquemático do circuito de conversão tempo-digital.

A.4 Interface histogramadora do Módulo TDC

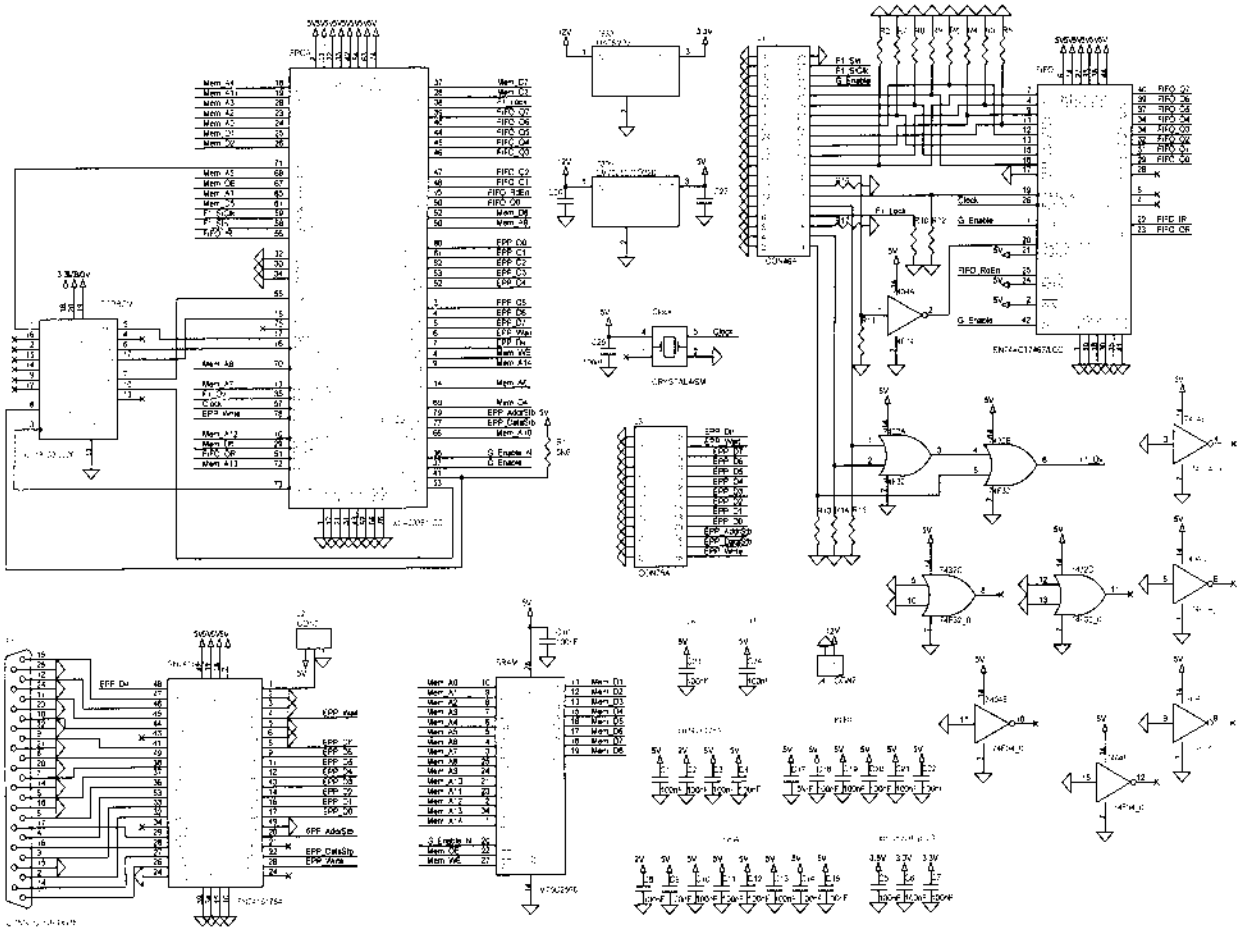

Figura A-4: Esquemático da interface histogramadora do *Módulo TDC*.

A.5 Conversor ADC

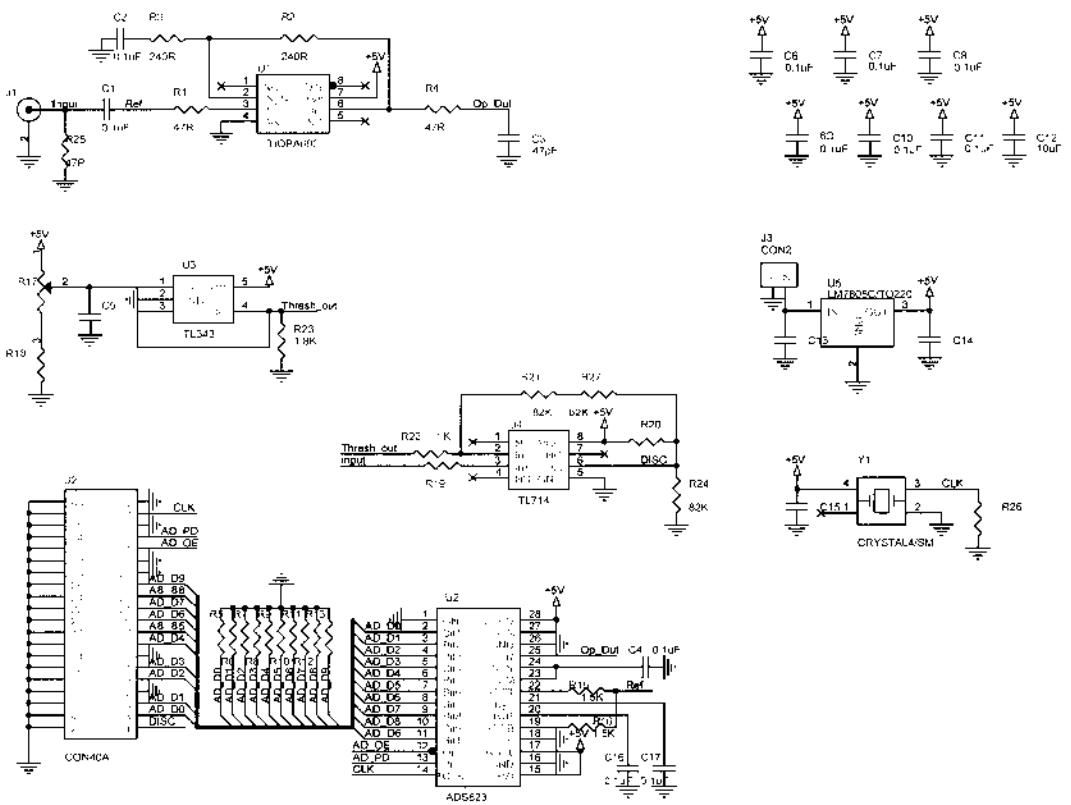

Figura A-5: Esquemático do circuito de conversão amplitude-digital.

A.6 Interface histogramadora do MCA

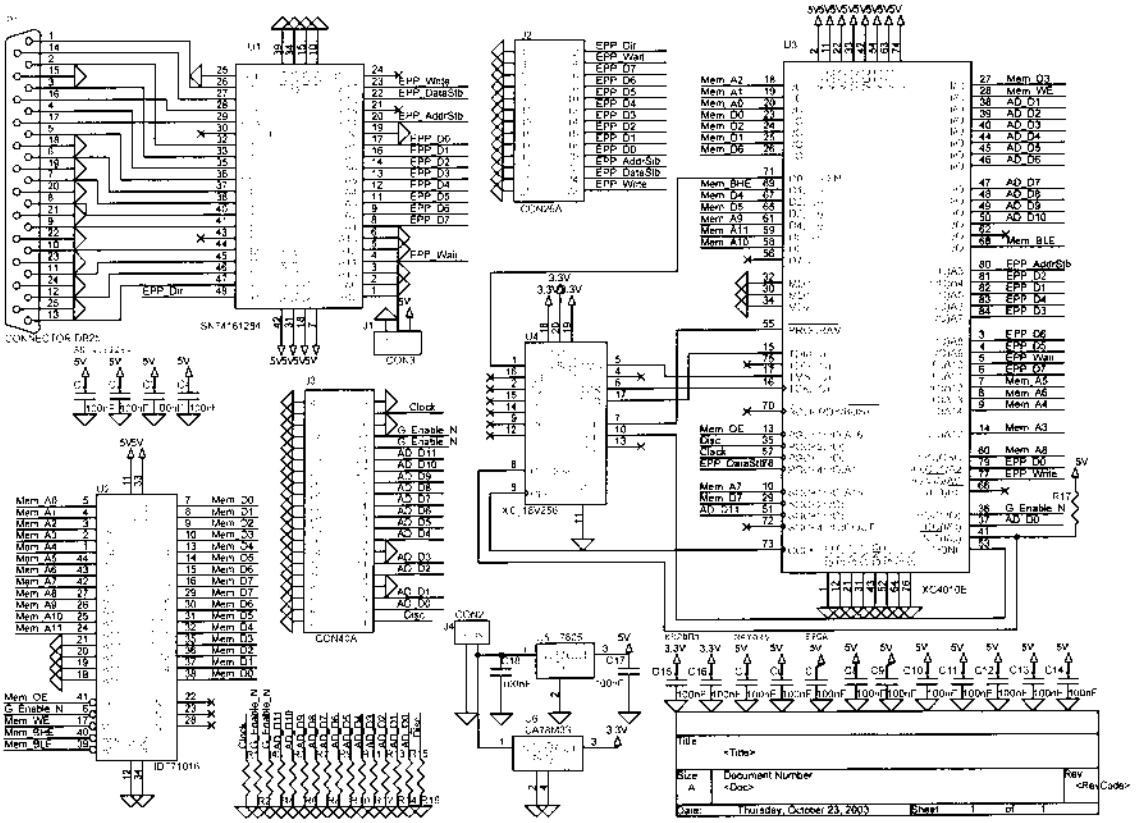

Figura A-6: Esquemático da interface histogramadora do *MCA*.

Apêndice B

Código das FPGAs

B.1 Módulo TDC

B.1.1 Esquemático do circuito interno à FPGA

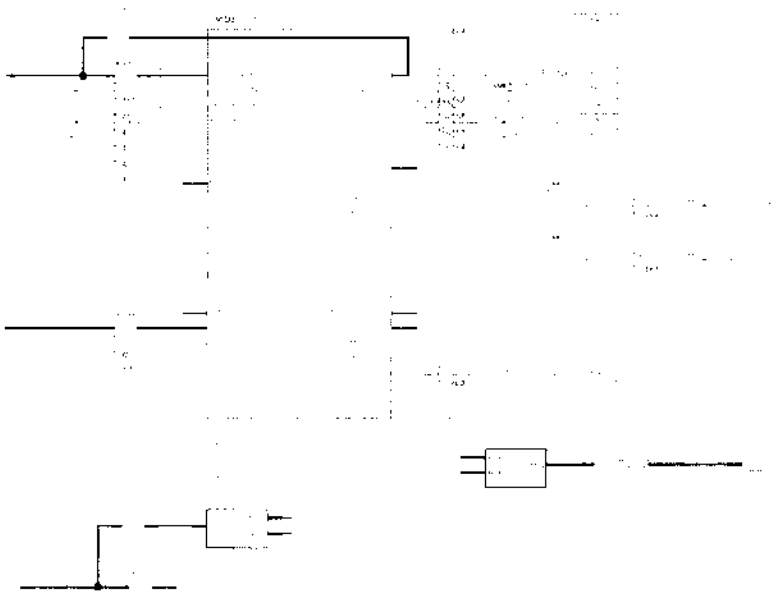

Figura B-1: Esquemático com a interligação dos blocos da FPGA do *Módulo TDC*.

B.1.2 Código em VHDL do bloco de Controle da EPP

```
entity PC_CTRL is
port (
EPP_Write: in STD_LOGIC;
EPP_DataStb: in STD_LOGIC;
EPP_AddrStb: in STD_LOGIC;
EPP_Wait: out STD_LOGIC;
EPP_Data_In: in STD_LOGIC_VECTOR (7 downto 0);
EPP_Data_Out: out STD_LOGIC_VECTOR (7 downto 0);
FIFO_IR: in STD_LOGIC;
F1_OF: in STD_LOGIC;
F1_Lock: in STD_LOGIC;
G_Enable: out STD_LOGIC;
F1_SIClk: out STD_LOGIC;
F1_SIN: out STD_LOGIC;
PC_Ctrl: out STD_LOGIC;
Sub: out STD_LOGIC;
Mem_OE_PC: out STD_LOGIC;
Mem_WE_PC: out STD_LOGIC;
Mem_Addr_PG: buffer STD_LOGIC_VECTOR (14 downto 0);
Mem_Data_In_PC: im STD_LOGIC_VECTOR (7 downto 0)
);
end PG_CTRL;

architecture PC_CTRL_arch of PG_CTRL is
signal Reset_Addr, Read_Status, G_Enable1, G_Enable2: STD_LOGIC;
begin
-- EPP Handshake

EPP_Wait <= (not EPP_AddrStb) or (not EPP_DataStb);

-- Controls

process (EPP_AddrStb) begin
if (EPP_AddrStb'event and EPP_AddrStb = '1') then
if EPP_Write = '0' then
   G_Enable1   <= EPP_Data_In(0);
   F1_SIClk    <= EPP_Data_In(1);
   F1_SIN      <= EPP_Data_In(2);
   PC_Ctrl     <= EPP_Data_In(3);
   Reset_Addr  <= EPP_Data_In(4);
   Read_Status <= EPP_Data_In(5);
   Sub         <= EPP_Data_In(6);
   G_Enable2   <= EPP_Data_In(7);
end if;
emd if;
end process;

G_Enable <= G_Enable1 and G_Enable2;

-- Memory access

Mem_OE_PC <= EPP_DataStb when EPP_Write = '1' else '1';
Mem_WE_PC <= EPP_DataStb when EPP_Write = '0' else '1';

-- Memory address generator

process (EPP_DataStb, Reset_Addr) begin
```

```
if Reset_Addr = '1' then
Mem_Addr_PC <= "111111111111111";
elsif (EPP_DataStb'event and EPP_DataStb = '0') then
Mem_Addr_PG <= Mem_Addr_PC + 1;
end if;
end process;

-- Data flow

EPP_Data_Dut(7 downto 4) <= Mem_Data_In_PG(7 downto 4);
EPP_Data_Out(3 downto 0) <= Mem_Data_In_PC(3 downto 0) when Read_Status = '0' else '0' & F1_Lock & F1_OF & FIFO_IR;
end PC_CTRL_arch;
```

B.1.3 Código em VHDL do Construtor de Endereços e Histogramador

```
entity Local_Ctrl is
port (
Clock: in STD_LOGIC;
Reset: in STD_LOGIG;
Pause: in STD_LOGIG;
Sub: in STD_LOGIG;
FIFO_DR: in STD_LOGIC;
FIFO_Q: in STD_LDGIC_VECTOR (7 downto 0);
FIFO_RdEn: out STD_LDGIC;
Idle: out STD_LOGIG;
Mem_OE_LC: out STD_LOGIC;
Mem_WE_LC: out STD_LOGIG;
Mem_Addr_LC: out STD_LOGIC_VEGTOR (14 downto 0);
Mem_Data_In_LC: in STD_LOGIC_VECTOR (7 downto 0);
Mem_Data_Out_LC: out STD_LDGIG_VECTOR (7 downto D)
);
end Local_Gtrl;

architecture Local_Ctrl_arch of Local_Ctrl is
type State1 is (A,B);
type State2 is (C,D,E);
type State3 is (F,G,H,I,J,L);
type State4 is (M,N);
type State5 is (O,P);
type State6 is (Q,R);
type State7 is (S,T);
signal S1: State1;
signal S2: State2;
signal S3: State3;
signal S4: State4;
signal S5: State5;
signal S6: State6;
signal S7: State7;
signal Empty_FIFD, Sync_Normal, Pause_In, Sync: STD_LOGIG;
signal Go_Suh_1, Go_Sub_2, Go_Sub_3, Go_Sub_4: STD_LOGIG;
signal Sync_Suh_1, Sync_Sub_2, Sync_Sub_3, Sync_Sub_4: STD_LOGIC;
signal Suh_1_Chan, Suh_2_Chan, Sub_3_Chan, Suh_4_Chan: STD_LOGIC;
signal Data_Suh_1, Data_Sub_2, Data_Sub_3, Data_Sub_4: STD_LOGIG_VEGTOR (13 downto 0);
signal Sub_1_Aux_1, Sub_1_Aux_2, Sub_2_Aux_1, Sub_2_Aux_2: STD_LOGIC_VECTDR (11 downto 0);
signal Sub_3_Aux_1, Sub_3_Aux_2, Sub_4_Aux_1, Sub_4_Aux_2: STD_LDGIG_VECTDR (11 downto D);
```

```
signal Oata_Aux, Mem_Addr_Normal: STD_LOGIC_VECTDR (13 downto 0);
begin
-- Empty_FIFD flag

process (FIFO_DR,Clock) begin
if FIFO_OR = '1' then
Empty_FIFD <= '1';
S1 <= A;
elsif (Clock'event and Clock = '1') then
case S1 is
   when A =>
       S1 <= B;
   when B =>
       Empty_FIFD <= 'D';
end case;
end if;
end process;

-- Assembly Data Normal

process (Empty_FIFD,Clock) begin
if Empty_FIFO = '0' then
S2 <= C;
Sync_Normal <= '0';
FIFO_RdEn <= '1';
elsif (Clock'event and Clock = '1') then
case S2 is
   when C =>
       S2 <= D;
       Data_Aux <= FIFO_Q(2 downto D) & Data_Aux(1D downto D);
       if Sync_Normal = '1' then
           Mem_Addr_Normal <= Data_Aux;
       end if;
       Sync_Normal <= '0';
   when D =>
       S2 <= E;
       Data_Aux <= Oata_Aux(13 downto 11) & FIFD_Q(4 downto D) & Data_Aux(5 downto D);
   when E =>
       if (Pause = '0' and Pause_In = '0') then
           S2 <= C;
           Sync_Normal <= '1';
           Data_Aux <= Data_Aux(13 downto 6) & FIFD_Q(7 downto 2);
       end if;
       FIFO_RdEn <= not (Pause or Pause_In);
end case;
end if;
end process;

-- Select Subtr Machine

Go_Sub_1 <= Sync_Normal when Data_Aux (13 downto 12) = "0D" else '0';
Go_Sub_2 <= Sync_Normal when Data_Aux (13 downto 12) = "01" else '0';
Co_Sub_3 <= Sync_Normal when Data_Aux (13 downto 12) = "10" else '0';
Go_Sub_4 <= Sync_Normal when Data_Aux (13 downto 12) = "11" else '0';

-- Assembly Oata Sub 1

process (Reset, Clock) begin
if Reset = '0' then
```

```
S4 <= M;
Sync_Sub_1 <= '0';
elsif (Clock'event and Clock = '1') then
case S4 is
   when M =>
       if Go_Sub_1 = '1' then
           S4 <= N;
           if Data_Aux (11) = '0' then
               Sub_1_Aux_1 <= '0' & Data_Aux (10 downto 0);
           else
               Sub_1_Aux_2 <= '0' & Data_Aux (10 downto 0);
           end if;
           Sub_1_Chan <= Data_Aux (11);
       end if;
       Sync_Sub_1 <= '0';
   when N =>
       if Go_Sub_1 = '1' then
           if Data_Aux (11) = Sub_1_Chan then
               if Data_Aux (11) = '0' then
                   Sub_1_Aux_1 <= '0' & Data_Aux (10 downto 0);
               else
                   Sub_1_Aux_2 <= '0' & Data_Aux (10 downto 0);
               end if;
               Sub_1_Chan <= Data_Aux (11);
               Sync_Sub_1 <= '0';
           else
               S4 <= M;
               Sync_Sub_1 <= '1';
               if Sub_1_Chan = '0' then
                   Data_Sub_1 <= "00" & (Sub_1_Aux_1 - Data_Aux (10 downto 0));
               else
                   Data_Sub_1 <= "00" & (Data_Aux (10 downto 0) - Sub_1_Aux_2);
               end if;
           end if;
       end if;
end case;
end if;
end process;

-- Assembly Data Sub 2

process (Reset, Clock) begin
if Reset = '0' then
S5 <= O;
Sync_Sub_2 <= '0';
elsif (Clock'event and Clock = '1') then
case S5 is
   when O =>
       if Go_Sub_2 = '1' then
           S5 <= P;
           if Data_Aux (11) = '0' then
               Sub_2_Aux_1 <= '0' & Data_Aux (10 downto 0);
           else
               Sub_2_Aux_2 <= '0' & Data_Aux (10 downto 0);
           end if;
           Sub_2_Chan <= Data_Aux (11);
       end if;
       Sync_Sub_2 <= '0';
   when P =>
```

```
        if Go_Sub_2 = '1' then
            if Data_Aux (11) = Sub_2_Chan then
                if Data_Aux (11) = '0' then
                    Sub_2_Aux_1 <= '0' & Data_Aux (10 downto 0);
                else
                    Sub_2_Aux_2 <= '0' & Data_Aux (10 downto 0);
                end if;
                Sub_2_Chan <= Data_Aux (11);
                Sync_Sub_2 <= '0';
            else
                S5 <= Q;
                Sync_Sub_2 <= '1';
                if Sub_2_Chan = '0' then
                    Data_Sub_2 <= "01" & (Sub_2_Aux_1 - Data_Aux (10 downto 0));
                else
                    Data_Suh_2 <= "01" & (Data_Aux (10 downto 0) - Sub_2_Aux_2);
                end if;
            end if;
        end if;
end case;
end if;
end process;

-- Assembly Data Sub 3

process (Reset, Clock) hegin
if Reset = '0' then
S6 <= Q;
Sync_Suh_3 <= '0';
elsif (Clock'event and Clock = '1') then
case S6 is
   when Q =>
        if Go_Sub_3 = '1' then
            S6 <= R;
            if Data_Aux (11) = '0' then
                Sub_3_Aux_1 <= 'D' & Data_Aux (10 downto 0);
            else
                Sub_3_Aux_2 <= 'D' & Data_Aux (10 downto 0);
            end if;
            Sub_3_Chan <= Data_Aux (11);
        end if;
        Sync_Sub_3 <= 'D';
   when R =>
        if Go_Sub_3 = '1' then
            if Data_Aux (11) = Sub_3_Chan then
                if Data_Aux (11) = '0' then
                    Suh_3_Aux_1 <= '0' & Data_Aux (1D downto 0);
                else
                    Sub_3_Aux_2 <= '0' & Data_Aux (1D downto 0);
                end if;
                Suh_3_Chan <= Data_Aux (11);
                Sync_Sub_3 <= 'D';
            else
                S6 <= Q;
                Sync_Sub_3 <= '1';
                if Sub_3_Chan = 'D' then
                    Data_Sub_3 <= "10" & (Suh_3_Aux_1 - Data_Aux (10 downto 0));
                else
                    Data_Suh_3 <= "1D" & (Data_Aux (1D downto 0) - Sub_3_Aux_2);
```

```
                end if;
            end if;
        end if;
end case;
end if;
end process;

-- Assembly Data Sub 4

process (Reset, Clock) begin
if Reset = '0' then
S7 <= S;
Sync_Sub_4 <= '0';
elsif (Clock'event and Clock = '1') then
case S7 is
   when S =>
        if Go_Sub_4 = '1' then
            S7 <= T;
            if Data_Aux (11) = 'D' then
                Sub_4_Aux_1 <= 'D' & Data_Aux (1D downto 0);
            else
                Sub_4_Aux_2 <= 'D' & Data_Aux (1D downto 0);
            end if;
            Sub_4_Chan <= Data_Aux (11);
        end if;
        Sync_Sub_4 <= '0';
   when T =>
        if Go_Sub_4 = '1' then
            if Data_Aux (11) = Sub_4_Chan then
                if Data_Aux (11) = 'D' then
                    Sub_4_Aux_1 <= '0' & Data_Aux (10 downto 0);
                else
                    Sub_4_Aux_2 <= '0' & Data_Aux (10 downto D);
                end if;
                Sub_4_Chan <= Data_Aux (11);
                Sync_Sub_4 <= '0';
            else
                S7 <= S;
                Sync_Sub_4 <= '1';
                if Sub_4_Chan = '0' then
                    Data_Sub_4 <= "11" & (Sub_4_Aux_1 - Data_Aux (10 downto 0));
                else
                    Data_Sub_4 <= "11" & (Data_Aux (1D downto 0) - Sub_4_Aux_2);
                end if;
            end if;
        end if;
end case;
end if;
end process;

-- Select Sync Signal

Sync <= Sync_Normal when Sub = '0'                      else
Sync_Sub_1  when Data_Aux (13 downto 12) = "0D" else
Sync_Sub_2  when Data_Aux (13 downto 12) = "01" else
Sync_Sub_3  when Data_Aux (13 downto 12) = "10" else
Sync_Sub_4;

-- Select Memory Address
```

```
Mem_Addr_LC (14 downto 1) <= Mem_Addr_Normal when Sub = '0'                    else
                    Data_Sub_1      when Data_Aux (13 downto 12) = "00" else
                    Data_Sub_2      when Data_Aux (13 downto 12) = "01" else
                    Data_Sub_3      when Data_Aux (13 downto 12) = "10" else
                    Data_Sub_4;

-- Histogram

process (Reset, Clock) begin
if Reset = 'D' then
S3 <= F;
Idle <= '0';
Mem_OE_LC <= '1';
Men_WE_LC <= '1';
Mem_Addr_LC(0) <= '0';
Pause_In <= '0';
elsif (Clock'event and Clock = '1') then
case S3 is
   when F =>
       if Sync = '0' then
           S3 <= F;
           Idle <= '1';
           Mem_OE_LC <= '1';
           Mem_WE_LC <= '1';
       else
           S3 <= G;
           Idle <= '0';
           Mem_OE_LC <= '0';
           Mem_WE_LC <= '1';
       end if;
       Mem_Addr_LC(0) <= '0';
   when G =>
       if Mem_Data_In_LC = "11111111" then
           S3 <= I;
           Pause_In <= '1';
       else
           S3 <= H;
           Pause_In <= '0';
       end if;
       Idle <= '0';
       Mem_OE_LC <= '1';
       Mem_WE_LC <= '1';
       Mem_Data_Out_LC <= Mem_Data_In_LC + 1;
   when H =>
       S3 <= P;
       Idle <= '0';
       Mem_OE_LC <= '1';
       Mem_WE_LC <= '0';
   when I =>
       S3 <= J;
       Idle <= 'D';
       Mem_OE_LC <= '1';
       Mem_WE_LC <= 'D';
   when J =>
       S3 <= L;
       Idle <= '0';
       Mem_OE_LC <= '0';
       Mem_WE_LC <= '1';
```

```
        Mem_Addr_LC(0) <= '1';
    when L =>
        S3 <= H;
        Idle <= 'D';
        Mem_OE_LC <= '1';
        Mem_WE_LC <= '1';
        Pause_In <= 'D';
        Mem_Data_Dut_LC <= Mem_Data_In_LC + 1;
end case;
end if;
end process;
end Local_Ctrl_arch;
```

B.2 MCA

B.2.1 Esquemático do circuito interno à FPGA

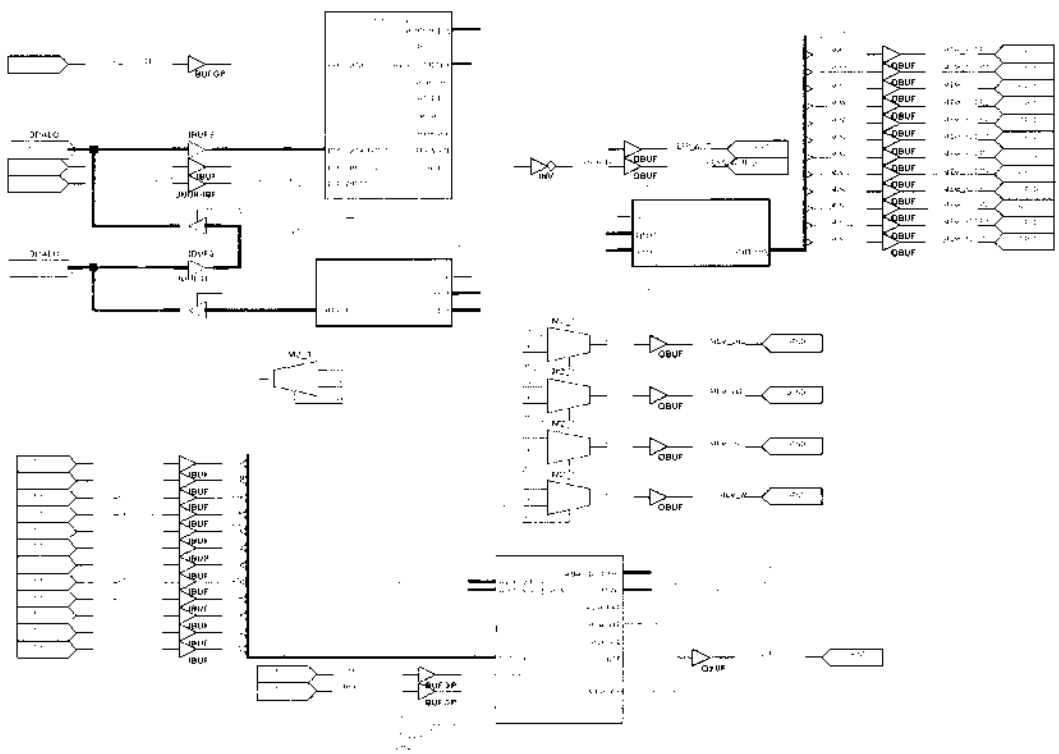

Figura B-2: Esquemático com interligação dos blocos internos à FPGA do MCA.

B.2.2 Código em VHDL do bloco de Controle da EPP

```
entity PC_Ctrl is
port (
EPP_AddrStb: in STD_LOGIC;
EPP_DataStb: in STD_LOGIC;
EPP_Write: in STD_LOGIC;
EPP_Wait: out STD_LOGIC;
G_Enable: out STD_LOGIC;
PC_Ctrl: out STD_LOGIC;
EPP_Data_In: in STD_LOGIC_VECTOR (7 downto 0);
Mem_OE: out STD_LOGIC;
Men_WE: out STD_LOGIC;
Mem_BLE: out STD_LOGIC;
Mem_BHE: out STD_LOGIC;
Mem_Addr: out STD_LOGIC_VECTOR (11 downto 0);
Window: out STD_LOGIC_VECTOR (3 downto 0)
);
end PC_Ctrl;

architecture PC_Ctrl_arch of PC_Ctrl is
signal Reset_Addr, G_Enable1, G_Enable2: STD_LOGIC;
signal Mem_Addr_In: STD_LOGIC_VECTOR (12 downto 0);
begin
-- Handshake

EPP_Wait <= (not EPP_AddrStb) or (not EPP_DataStb);

-- Controls

process (EPP_AddrStb) begin
if (EPP_AddrStb'event and EPP_AddrStb = '1') then
if EPP_Write = '0' then
   G_Enable1  <= EPP_Data_In(0);
   Window(0)  <= EPP_Data_In(1);
   Window(1)  <= EPP_Data_In(2);
   PC_Ctrl    <= EPP_Data_In(3);
   Reset_Addr <= EPP_Data_In(4);
   Window(2)  <= EPP_Data_In(5);
   Window(3)  <= EPP_Data_In(6);
   G_Enable2  <= EPP_Data_In(7);
end if;
end if;
end process;

G_Enable <= G_Enable1 and (not G_Enable2);

-- Memory access

Mem_OE <= EPP_DataStb when EPP_Write = '1' else '1';
Mem_WE <= EPP_DataStb when EPP_Write = '0' else '1';

-- Memory address generator

process (EPP_DataStb, Reset_Addr) begin
if Reset_Addr = '1' then
Mem_Addr_In <= "1111111111111";
elsif (EPP_DataStb'event and EPP_DataStb = '0') then
Mem_Addr_In <= Mem_Addr_In + 1;
```

```
end if;
end process;

Mem_BLE <= Mem_Addr_In(0);
Mem_BHE <= not Mem_Addr_In(0);

Mem_Addr <= Mem_Addr_In(12 downto 1);
end PC_Ctrl_arch;
```

B.2.3 Código em VHDL do Integrador e do Histogramador

```
entity Local_Ctrl is
port (
Rst : in STD_LOGIC;
Clock : in STD_LOGIC;
Data : in STD_LOGIC_VEGTOR (11 downto 0);
Disc : in STD_LOGIC;
Outp : buffer STD_LOGIC;
Mem_Addr: out STD_LOGIC_VEGTOR (11 downto 0);
Mem_OE: out STD_LGGIC;
Mem_WE: out STD_LOGIC;
Mem_BLE: out STD_LOGIG;
Mem_BHE: out STD_LOGIC;
Mem_Data_In: in STD_LOGIC_VECTOR (7 downto 0);
Mem_Data_Out: out STD_LOGIC_VECTOR (7 downto 0);
Window: in STD_LDGIC_VECTOR (3 downto 0)
);
end Local_Ctrl;

architecture Local_Ctrl_arch of Local_Ctrl is
type State3 is (F,G,H,I,J,L);
signal S3: State3;
signal aux      : STO_LOGIC_VECTOR (11 downto 0);
signal Disc_int  : STD_LOGIC;
signal Disc_rst  : STD_LOGIC;
signal Disc_cont  : STD_LOGIC_VECTOR (3 downto 0);
signal Addr: STD_LOGIC_VECTOR (9 downto D);
begin
Process(Rst,Clock)
begin
if Rst = '1' then
Disc_cont <= "ODDD";
elsif (Clock'event and Clock = '1') then
if Oisc_int = '1' then
   Disc_cont <= Disc_cont + 1;
end if;
if Disc_cont = Window then
   Disc_rst <= '1';
   Outp <= '1';
   Disc_cont <= "0000";
   Mem_Addr <= aux;
else
   Outp <= 'O';
   Disc_rst <= 'O';
end if;
end if;
end Process;
```

```
--
Process(Rst,Disc,Disc_rst)
begin
if Rst = '1' or Disc_rst = '1' then
Disc_int <= '0';
elsif (Disc'event and Disc = '1') then
Disc_int <= '1';
end if;
end Process;
--
Process(Rst,Clock,Disc_rst)
begin
if Rst = '1' or Disc_rst = '1' then
aux <= "000000000000";
elsif (Clock'event and Clock = '1') then
--if Disc_int = '1' then
   if Data > aux then
       aux <= Data;
   else
       aux <= aux;
   end if;
--end if;
end if;
end Process;

-- Histogran

process (Rst, Clock) begin
if Rst = '1' then
S3 <= F;
Mem_OE <= '1';
Mem_WE <= '1';
Mem_BLE <= '0';
Mem_BHE <= '1';
elsif (Clock'event and Clock = '1') then
case S3 is
   when F =>
       if Outp = '0' then
           S3 <= F;
           Mem_OE <= '1';
           Mem_WE <= '1';
       else
           S3 <= G;
           Mem_OE <= '0';
           Mem_WE <= '1';
       end if;
       Mem_BLE <= '0';
       Mem_BHE <= '1';
   when C =>
       if Mem_Data_In = "11111111" then
           S3 <= I;
       else
           S3 <= H;
       end if;
       Hem_OE <= '1';
       Mem_WE <= '1';
       Mem_Oata_Out <= Mem_Data_In + 1;
   when H =>
       S3 <= F;
```

```
        Mem_OE <= '1';
        Mem_WE <= '0';
    when I =>
        S3 <= J;
        Mem_OE <= '1';
        Mem_WE <= '0';
    when J =>
        S3 <= L;
        Mem_OE <= '0';
        Mem_WE <= '1';
        Mem_BLE <= '1';
        Mem_BHE <= '0';
    when L =>
        S3 <= H;
        Mem_OE <= '1';
        Mem_WE <= '1';
        Mem_Data_Out <= Mem_Data_In + 1;
end case;
end if;
end process;
end Local_Ctrl_arch;
```

Apêndice C

Código do programa Chambers 1.0

C.1 A classe TF1

```
unit F1;

interface

uses
Classes;

type
TF1 = class;

TDACType = (dt8841,dt8842);

TDAChannel = class(TPersistent)
private
FRefVoltage: Real;
FVoltage: Real;
FF1: TF1;
SVoltage: Real;
SRegValue: Byte;
FOnChangeRefVoltage: TNotifyEvent;
FOnChangeVoltage: TNotifyEvent;
FOnChangeValue: TNotifyEvent;
function GetRegValue: Byte;
function GetMinValue: Byte;
function GetMaxValue: Byte;
function GetMinVoltage: Real;
function CetMaxVoltage: Real;
procedure ChangeRefVoltage;
procedure ChangeVoltage;
procedure ChangeValue;
procedure SetRefVoltage(Value: Real);
procedure SetVoltage(Value: Real);
public
constructor Create(F1: TF1); reintroduce;
function ValueToVoltage(AValue: Byte): Real;
```

```
function VoltageToValue(AVoltage: Real): Byte;
function NextVoltageUp: Real;
function NextVoltageDown: Real;
procedure Update;
property RegValue: Byte read GetRegValue;
property MinValue: Byte read GetMinValue;
property MaxValue: Byte read GetMaxValue;
property MinVoltage: Real read GetMinVoltage;
property MaxVoltage: Real read GetMaxVoltage;
property OnChangeRefVoltage: TNotifyEvent read FOnChangeRefVoltage write FOnChangeRefVoltage;
property OnChangeVoltage: TNotifyEvent read FOnChangeVoltage write FOnChangeVoltage;
property OnChangeValue: TNotifyEvent read FOnGhangeValue write FOnChangeValue;
published
property RefVoltage: Real read FRefVoltage write SetRefVoltage;
property Voltage: Real read FVoltage write SetVoltage;
end;

TInput = class(TPersistent)
private
FSingleEnded: Boolean;
FDAChannel: TDAChannel;
procedure SetSingleEnded(Value: Boolean);
public
constructor Greate(AOwner: TF1); reintroduce;
destructor Destroy; override;
published
property SingleEnded: Boolean read FSingleEnded write SetSingleEnded default True;
property DAChannel: TDAChannel read FDAGhannel write FDAChannel;
end;

TStart = class(TInput);

TStop = class(TInput)
privaie
FEnabled: Boolean;
FRiseTrig: Boolean;
FFallTrig: Boolean;
FDnChangeEnabled: TNotifyEvent;
FDnChangeRiseTrig: TNotifyEvent;
FDnGhangeFallTrig: TNotifyEvent;
procedure ChangeEnabled;
procedure ChangeRiseTrig;
procedure ChangeFalltrig;
procedure SetEnahled(Value: Boolean);
procedure SetRiseTrig(Value: Boolean);
procedure SetFallTrig(Value: Boolean);
public
constructor Create(ADwner: TF1);
property OnChangeEnahled: TNotifyEvent read FDnChangeEnahled write FDnChangeEnabled;
property DnChangeRiseTrig: TNotifyEvent read FDnChangeRiseTrig write FDnChangeRiseTrig;
property DnChangeFallTrig: TNotifyEvent read FDnGhangeFallTrig write FonChangeFallTrig;
published
property Enahled: Boolean read FEnabled write SetEnabled default True;
property RiseTrig: Boolean read FRiseTrig write SetRiseTrig default True;
property FallTrig: Boolean read FFallTRig write SetFallTrig default False;
end;

TPLL = class(TPersistent)
private
```

```
FDwner: TF1;
FRefDivider: Byte;
FClockDivider: Byte;
FOnChangeRefDivider: TNotifyEvent;
FDnChangeClockDivider: TNotifyEvent;
function GetPLLValue: Real;
procedure ChangeRefDivider;
procedure GhangeClockDivider;
procedure SetRefDivider(Value: Byte);
procedure SetGlockDivider(Value: Byte);
public
constructor Create(AOwner: TF1); reintroduce;
property PLLValue: Real read GetPLLValue;
property DnChageClockDivider: TNotifyEvent read FOnGhangeClockDivider write FDnChangeGlockDivider;
property DnChangeRefDivider: TNotifyEvent read FDnChangeRefDivider write FDnGhangeRefDivider;
published
property RefDivider: Byte read FRefDivider write SetRefDivider default 0;
property ClockDivider: Byte read FClockDivider write SetClockDivider default 1;
end;

TDataBits = (db24Bits, dbBBits);
TF1Mode = (fmTrigger, fmCommon);

TF1 = class(TComponent)
private
FEnabledStops: Boolean;
FHighResolution: Boolean;
FClockFreq: Real;
FPLL: TPLL;
FStsrt: TStart;
FStop1: TStop;
FStop2: TStop;
FStop3: TStop;
FStop4: TStop;
FStop5: TStop;
FStop6: TStop;
FStop7: TStop;
FStop8: TStop;
FDAGType: TDACType;
FConfDAC: Boolean;
FDstaBits: TDataBits;
FF1Mode: TF1Mode;
function GetMisTime: Real;
function GetMaxTime: Real;
function GetResolution: Real;
procedure SetDAGType(Value: TDAGType);
procedure SetEnahledStops(Value: Bnolean);
procedure SetHighResolution(Value: Boolean);
procedure SetClockFreq(Value: Real);
public
Stops: array [1..B] of TStop;
constructor Greate(ADwner: TComponent); override;
destructor Destroy; override;
property Resolution: Real read GetResolution;{picosecounds}
property MinTime: Real read GetMinTime;
property MaxTime: Real read GetMaxTime;
property F1Mode: TF1Mode read FF1Mode write FF1Mode;
property DataBits: TDataBits read FDataBits write FDataBits;
published
```

```
property EnableStops: Boolean read FEnabledStops write SetEnabledStops default True;
property HighResolution: Boolean read FHighResolution write SetHighResolution default False;
property ClockFreq: Real read FClockFreq write SetClockFreq; {MHz}
property Start: TStart read FStart write FStart;
property Stop1: TStop read FStop1 write FStop1;
property Stop2: TStop read FStop2 write FStop2;
property Stop3: TStop read FStop3 write FStop3;
property Stop4: TStop read FStop4 write FStop4;
property Stop5: TStop read FStop5 write FStop5;
property Stop6: TStop read FStop6 write FStop6;
property Stop7: TStop read FStop7 write FStop7;
property Stop8: TStop read FStop8 write FStop8;
property DACType: TDACType read FDACType write SetDACType default dt8841;
property PLL: TPLL read FPLL write FPLL;
end;

procedure Register;

implementation

{$R *.dcr}

uses
Math, IOControl, Windows;

procedure Register;
begin
RegisterComponents('My Components', [TF1]);
end;

//TDAChannel

constructor TDAChannel.Create(F1: TF1);
begin
FVoltage := 1.4;{Volts}
FRefVoltage := 1.5;{Volts}
FF1 := F1;
Update;
end;

function TDAChannel.ValueToVoltage(AValue: Byte): Real;
begin
if FF1.DACType = dt8B41 then
Result := AValue/128*FRefVoltage
else Result := (AValue/12S -1)*FRefVoltage;
end;

function TDAChannel.VoltageToValue(AVoltage: Real): Byte;
begin
if FF1.DACType = dt8841 then
Result := Round((AVoltage/FRefVoltage)*128)
else Result := Round((AVoltage/FRefVoltage + 1)*128);
end;

function TDAChannel.GetRegValue: Byte;
begin
Result := VoltageToValue(FVoltage);
end;
```

```
function TDAChannel.GetMinValue: Byte;
begin
Result := VoltageToValue(0.8);
if ValueToVoltage(Result) < 0.8 then Result := Result+1;
end;

function TDAChannel.GetMaxValue: Byte;
begin
Result := VoltageToValue(2.8);
if ValueToVoltage(Result) > 2.8 then Result := Result-1;
end;

function TDAChannel.GetMinVoltage: Real;
begin
Result := ValueToVoltage(MinValue);
end;

function TDAChannel.GetMaxVoltage: Real;
begin
Result := ValueToVoltage(MaxValue);
end;

function TDAChannel.NextVoltageUp: Real;
begin
Voltage := ValueToVoltage(RegValue+1);
Result := Voltage;
end;

function TDAChannel.NextVoltageDown: Real;
begin
Voltage := ValueToVoltage(RegValue-1);
Result := Voltage;
end;

procedure TDAChannel.ChangeRefVoltage;
begin
if Assigned(OnChangeRefVoltage) then OnChangeRefVoltage(Self);
end;

procedure TDAChannel.CbangeVoltage;
begin
if Assigned(OnChangeVoltage) then OnChangeVoltage(Self);
end;

procedure TDAChannel.ChangeValue;
begin
if Assigned(OnChangeValue) then OnChangeValue(Self);
end;

procedure TDAChannel.Update;
begin
FVoltage := ValueToVoltage(RegValue);
if (SVoltage <> FVoltage) then ChangeVoltage;
if (SRegValue <> RegValue) then ChangeValue;
SRegValue := RegValue;
SVoltage := FVoltage;
end;

procedure TDAChannel.SetRefVoltage(Value: real);
```

```
begin
if Value = FRefVoltage then Exit;
SRegValue := RegValue;
SVoltage := FVoltage;
FRefVoltage := Value;
Update;
ChangeRefVoltage;
end;

procedure TDAChannel.SetVoltage(Value: Real);
begin
if (Value = FVoltage) or (Value < MinVoltage) or (Value > MaxVoltage) then Exit;
SRegValue := RegValue;
SVoltage := FVoltage;
FVoltage := Value;
Update;
end;

//TInput

constructor TInput.Create(AOwner: TF1);
begin
FSingleEnded := True;
FDAChannel := TDAChannel.Create(AOwner);
end;

destructor TInput.Destroy;
begin
FDAChannel.Free;
inherited;
end;

procedure TInput.SetSingleEnded(Value: Boolean);
begin
FSingleEnded := True;
end;

//TStop

constructor TStop.Create(AOwner: TF1);
begin
inherited Create(AOwner);
FEnabled := True;
FRiseTrig := True;
FFallTrig := False;
end;

procedure TStop.ChangeEnabled;
begin
if Assigned(OnChangeEnabled) then OnChangeEnabled(Self);
end;

procedure TStop.ChangeRiseTrig;
begin
if Assigned(OnChangeRiseTrig) then OnChangeRiseTrig(Self);
end;

procedure TStop.ChangeFallTrig;
begin
```

```
if Assigned(DnChangeFallTrig) then OnChangeFallTrig(Self);
end;

procedure TStop.SetEnabled(Value: Boolean);
begin
if Value = FEnabled then Exit;
FEnabled := Value;
if not FEnabled then begin
RiseTrig := False;
FallTrig := False;
end;
ChangeEnabled;
end;

procedure TStop.SetRiseTrig(Value: Boolean);
begin
if Value = fRiseTrig then Exit;
FRiseTrig := Value;
ChangeRiseTrig;
end;

procedure TStop.SetFallTrig(Value: Boolean);
begin
if Value = FFallTrig then Exit;
FFallTrig := Value;
ChangeFallTrig;
end;

//TPLL

constructor TPLL.Create(AOwner: TF1);
begin
inherited Create;
FOwner := AOwner;
FRefDivider := 0;
FClockDivider := 1;
end;

function TPLL.GetPLLValue: Real;{picosecounds}
begin
Result := ((1000000/FOwner.ClockFreq)*Power(2,RefDivider))/(152*ClockDivider);
end;

procedure TPLL.ChangeRefDivider;
begin
if Assigned(FOnChangeRefDivider) then FOnChangeRefDivider(Self);
end;

procedure TPLL.ChangeClockDivider;
begin
if Assigned(FOnChangeClockDivider) then FOnChangeClockDivider(Self);
end;

procedure TPLL.SetRefDivider(Value: Byte);
begin
if (Value = FRefDivider) or (Value > 7) then Exit;
FRefDivider := Value;
ChangeRefDivider;
end;
```

```
procedure TPLL.SetClockDivider(Value: Byte);
begin
if (Value = FClockDivider) then Exit;
FClockDivider := Value;
ChangeClockDivider;
end;

//TF1

constructor TF1.Create(AOwner: TComponent);
begin
inherited Create(ADwner);
FDataBits := db8Bits;
FF1Mode := fmCommon;
FClockFreq := 1;{MHz}
FPLL := TPLL.Create(Self);
FEnahledStops := True;
FHighResolution := False;
FStart := TStart.Create(Self);
FStop1 := TStop.Create(Self);
Stops[1] := FStop1;
FStop2 := TStop.Create(Self);
Stops[2] := FStop2;
FStop3 := TStop.Create(Self);
Stops[3] := FStop3;
FStop4 := TStop.Create(Self);
Stops[4] := FStop4;
FStop5 := TStop.Create(Self);
Stops[5] := FStop5;
FStop6 := TStop.Create(Self);
Stops[6] := FStop6;
FStop7 := TStop.Create(Self);
Stops[7] := FStop7;
FStop8 := TStop.Create(Self);
Stops[8] := FStop8;
end;

destructor TF1.Deetroy;
begin
FStart.Free;
FStop1.Free;
FStop2.Free;
FStop3.Free;
FStop4.Free;
FStop5.Free;
FStop6.Free;
FStop7.Free;
FStop8.Free;
FPLL.Free;
inherited Destroy;
end;

function TF1.GetResolution: Real;
begin
Result := FPLL.PLLValue;
if HighResolution then Result := Result/2;
end;
```

```
function TF1.GetMinTime: Real;
begin
Result := 5;{ns}
end;

function TF1.GetMaxTime: Real;{ns}
begin
Result := Resolution*$FFFF/1000;
end;

procedure TF1.SetDACType(Value: TDACType);
begin
if Value = FDACType then Exit;
FDACType := Value;
Start.FDAChannel.Update;
Stop1.FDAChannel.Update;
Stop2.FDAChannel.Update;
Stop3.FDAChannel.Update;
Stop4.FDAChannel.Update;
Stop5.FDAChannel.Update;
Stop6.FDAChannel.Update;
Stop7.FDAChannel.Update;
Stop8.FDAChannel.Update;
end;

procedure TF1.SetEnabledStops(Value: Boolean);
begin
if Value = FEnabledStops then Exit;
FEnabledStops := Value;
end;

procedure TF1.SetHighResolution(Value: Boolean);
begin
if Value = FHighResolution then Exit;
FHighResolution := Value;
end;

procedure TF1.SetClockFreq(Value: Real);
begin
if Value = FClockFreq then Exit;
FClockFreq := Value;
end;

end.
```

C.2 A classe TF1Parallel

```
unit F1Parallel;

interface

uses
Classes, F1, Graphics;

type
TMemControl = (mcPC, mcLocal);

TReadMode = (rmMemory, rmStatus);

TProgEvent = procedure(Sender: TObject; Stage: TProgressStage; PercentDone,ClkDiv,RefDiv: Byte) of object;

TF1Parallel = class(TComponent)
private
FF1: TF1;
FEnabled: Boolean;
FControls: Byte;
FMemControl: TMemControl;
FConfDAC: Boolean;
FRunning: Boolean;
FReadMode: TReadMode;
FRefDiv: Byte;
FDnScanResProg: TProgEvent;
FSubtraction: Boolean;
FCanCheckFlags: Boolean;
function GetResolution: Real;
function GetChannels: Word;
functiou GetMaxTime: Real;
function GetHardwareExists: Boolean;
function GetPLLIsLocked: Boolean;
function GetFIFDDverFlow: Boolean;
function GetF1OverFlow: Boolean;
procedure WRegisterData(Reg: Byte);
procedure Configure(Add: Byte; Data: Word);
procedure ResetMemAddr;
procedure Prog(Sender: TDbject; Stage: TProgressStage; PercentDone, ClkDiv, RefDiv: Byte);
procedure SetEnabled(Value: Boolean);
procedure SetMemControl(Value: TMemControl);
procedure SetReadMode(Value: TReadMode);
procedure SetRefDiv(Value: Byte);
procedure SetSubtraction(Value: Boolean);
procedure SetCanCheckFlags(Value: Boolean);
property Enahled: Boolean read FEnabled write SetEnahled;
property MemControl: TMemControl read FMemControl write SetMemControl;
property ReadMode: TReadMode read FreadMode write SetReadMode;
public
Stops: array of array of Integer;
constructor Create(AOwner: TComponent); override;
procedure ReadMemory;
procedure ClearMemory;
procedure Start;
procedure Stop;
procedure ScanResolution(var AClockDivider, ARefDivider: array of Byte; var Length: Integer);
property Running: Boolean read FRunning;
property PLLIsLocked: Boolean read GetPLLIsLocked;
```

```
property Resolution: Real read GetResolution;
property Channels: Word read GetChannels;
property MaxTime: Real read GetMaxTime;
property HardwareExists: Boolean read GetHardwareExists;
property FIFOOverFlow: Boolean read GetFIFOOverFlow;
property F1OverFlow: Boolean read GetF1OverFlow;
property CanCheckFlags: Boolean read fCanCheckFlags write SstCanCheckFlags;
published
property F1: TF1 read FF1 write FF1;
property RefDiv: Byte read FRefOiv write SetRefDiv default 2;
property Subtraction: Boolean read FSuhtraction write SetSuhtraction default False;
property OnScanResProg: TProgEvent read FOnScanResProg write FOnScanResProg;
end;

procednre Register;

implementation

uses
IOControl, Windows, Math;

procedure Register;
begin
RegisterComponents('My Components', [TF1Parallel]);
end;

// TF1Parallel

constructor TF1Parallel.Create(AOwner: TComponent);
begin
inherited Create(AOwner);
FRefDiv := 2;
SetLength(Stops,B,Channels);
output_hyte($37A,0);
fCanCheckFlags := True;
end;

function TF1Parallel.GetPLLIsLocked: Boolean;
begin
if fCanCheckFlags then begin
ReadMode := rmStatus;
Result := (input_byte($37B) and $04 <> 0) and HardwareExists;
ReadMode := rmMemory; end
else Result := False;
end;

procedure TF1Parallel.Configure(Add: Byte; Data: Word);
procedure SendBit(Bit: Boolean);
begin
if Bit then begin
FControls := FControls or $04;
output_byte($37B,FControls);end
else begin
FControls := FControls and $FB;
output_byte($37B,FControls);
end;
end;
procedure Clock;
var
```

```
i: Integer;
begin
for i := 0 to 3 do begin
FControls := FControls or $02;
output_byte($37B,FControls);
FControls := FControls and $FD;
output_byte($37B,FControls);
end;
end;
var
i: Integer;
begin
//start bit
SendBit(True);
Clock;
SendBit(False);
Clock;
//addr
for i := 0 to 7 do begin
SendBit((Add and $80)<>0);
Clock;
Add := Add*2;
end;
//data
for i := 0 to 15 do begin
SendBit((Data and $8000)<>0);
Clock;
Data := Data*2;
end;
//stop bit
SendBit(True);
Clock;
SendBit(False);
Clock;
end;

procedure TF1Parallel.ResetMemAddr;
begin
FControls := FControls or $10;
output_byte($37b,FControls);
FControls := FControls and $EF;
output_byte($37b,FControls);
end;

procedure TF1Parallel.ReadMemory;
var
i,j,Chan: Integer;
SEnabled: Boolean;
SMemControl: TMemControl;
begin
if not HardwareExists then Exit;
ReadMode := rmMemory;
SEnabled := Enabled;
Enabled := True;
SMemControl := MemControl;
MemControl := mcPC;
if FSubtraction then begin
for i := 0 to 3 do
for j := D to Channels-1 do
```

```
Stops[i,j] := 0; end
else
for i := 0 to 7 do
for j := 0 to Channels-1 do
Stops[i,j] := 0;
ResetMemAddr;
if not FSubtraction then begin
for i := 0 to 7 do
for j := 0 to 2047 do begin
Chan := Trunc(j/Power(2,FRefDiv-2));
Stops[i,Chan] := Stops[i,Chan] + input_byte($37C);
Stops[i,Chan] := Stops[i,Chan] + 256*input_byte($37C);
end;end
else begin
for i := 0 to 7 do
for j := 0 to 2047 do begin
Chan := (Channels div 2)*Integer(not Odd(i))+Trunc(j/Power(2,FRefDiv-2));
Stops[i div 2,Chan] := Stops[i div 2,Chan]+ input_byte($37C);
Stops[i div 2,Chan] := Stops[i div 2,Chan]+ 256*input_byte($37C);
end;
end;

MemControl := SMemControl;
Enabled := SEnabled;
end;

procedure TF1Parallel.ClearMemory;
var
i: Integer;
SEnabled: Boolean;
SMemControl: TMemControl;
begin
SEnabled := Enabled;
Enabled := True;
SMemControl := MemControl;
MemControl := mcPC;
ReadMode := rmMemory;
ResetMemAddr;
for i := 0 to 32*1024-1 do output_byte($37C,0);
MemControl := SMemControl;
Enabled := SEnabled;
end;

procedure TF1Parallel.SetEnabled(Value: Boolean);
begin
FEnabled := Value;
if FEnabled then FControls := FControls or $B1 else FControls := FControls and $7E;
output_byte($37b,FControls);
end;

procedure TF1Parallel.SetMemControl(Value: TMemControl);
begin
FMemControl := Value;
if FMemControl = mcPC then FControls := FControls or $0B else FControls := FControls and $F7;
output_byte($37b,FControls);
end;

procedure TF1Parallel.Start;
begin
```

```
FRunning := True;
Enabled := True;
MemControl := mcLocal;
WRegisterData(15);//start ring oscillator, f1 mode e bus width
WRegisterData(10);//set resolution
WRegisterData(1);//high or normal resolution
WRegisterData(2);//set edges
WRegisterData(3);
WRegisterData(4);
WRegisterData(5);

WRegisterData(11);//set threshold
WRegisterData(11);
FConfDAC := True;
WRegisterData(1);//configure DAC
WRegisterData(1);
WRegisterData(12);//set threshold
WRegisterData(12);
FConfDAC := True;
WRegisterData(1);//configure DAC
WRegisterData(1);
WRegisterData(13);//set threshold
WRegisterData(13);
FConfDAC := True;
WRegisterData(1);//configure DAC
WRegisterData(1);
WRegisterData(14);//set threshold
WRegisterData(14);
FConfDAC := True;
WRegisterData(1);//configure DAC
WRegisterData(1);
WRegisterData(11);//set threshold
WRegisterData(11);
FConfDAC := True;
WRegisterData(1);//configure DAC
WRegisterData(1);
WRegisterData(12);//set threshold
WRegisterData(12);
FConfDAC := True;
WRegisterData(1);//configure DAC
WRegisterData(1);
WRegisterData(13);//set threshold
WRegisterData(13);
FConfDAC := True;
WRegisterData(1);//configure DAC
WRegisterData(1);
WRegisterData(14);//set threshold
WRegisterData(14);
FConfDAC := True;
WRegisterData(1);//configure DAC
WRegisterData(1);

FConfDAC := False;
WRegisterData(7);
end;

procedure TF1Parallel.Stop;
begin
MemControl := mcPC;
```

```
Enabled := False;
FRunning := False;
end;

procedure TF1Parallel.WRegisterData(Reg: Byte);
var
Data: Word;
begin
with FF1 do case Reg of
1: Data := Byte(HighResolutinn)*$B0D0 + Byte(FConfDAC);
2: Data := Byte(Stop2.FallTrig)*$BDD0 + Byte(Stop2.RiseTrig)*$4D00 + $3fD0 + Byte(Stop1.FallTrig)*128 +
       Byte(Stop1.RiseTrig)*64 + $DD3f;
3: Data := Byte(Stop4.FallTrig)*$BDDD + Byte(Stop4.RiseTrig)*$40D0 + $3f00 + Byte(Stop3.FallTrig)*128 +
       Byte(Stop3.RiseTrig)*64 + $0D3f;
4: Data := Byte(Stop6.FallTrig)*$B0DD + Byte(Stop6.RiseTrig)*$4000 + $3f00 + Byte(Stop5.FallTrig)+128 +
       Byte(Stop5.RiseTrig)*64 + $0D3f;
5: Data := Byte(StopB.FallTrig)*$B00D + Byte(StopB.RiseTrig)*$4D0D + $3fDD + Byte(Stop7.Fal1Trig)*128 +
      Byte(Stop7.RiseTrig)*64 + $D03f;
7: Data := $8DDD;
10: Data := $1B00 + PLL.RefDivider*256 + PLL.ClockDivider;
1I: Data := Stop1.DAChannel.RegValue*256 + Start.DAChannel.RegValue;
12: Data := Stop3.DAChannel.RegValue*256 + Stop2.DAChannel.RegVaiue;
13: Data := 5top5.DAChanneI.RegValue*256 + Stop4.DAChannel.RegValue;
14: Data := Stop7.DAChannel.RegValue*256 + Stop6.DAChannel.RegValue;
15: Data := 8 + Byte(F1Mode)*2 + Byte(DataBits);
else Data := D;
end;
Configure(Reg,Data);
end;

procedure TF1Parallel.ScanResolution(var AClockDivider, ARefDivider: array of Byte; var Length: Integer);
var
ClockDivider,RefDivider: Byte;
SClockDivider,SRefDivider: Byte;
Index, AuxClkDiv,AuxRefDiv,i,j: Integer;
Res: array of Real;
AuxRes: Real;
Aux: Boolean;
begin
if not HardwareExists then Exit;
SetLength(Res,2D4S);
SClockDivider := FF1.PLL.ClockDivider;
SRefDivider := FF1.PLL.RefDivider;
Index := D;
j := D;
Prog(Self,psStarting,D,0,D);
for RefDivider := D to 7 do begin
FF1.PLL.RefDivider := RefDivider;
for ClockDivider := 1 to 255 do begin
FF1.PLL.ClockDivider := ClnckDivider;
Aux := False;
for i := 0 to Index do if Round(FF1.Resolution) = Round(Res[i]) then Aux := True;
if (FF1.Resolution > 1D0) and (FFI.Resolution < 20D) and not Aux then begin
Start;
Sleep(I0D);
if PLLIsLocked then begin
ARefDivider[Index] := RefDivider;
AClnckDivider[Index] := ClockDivider;
Res[Index] := FF1.Resolution;
```

```
Inc(Index);
end;
Stop;
end;
if (j mod 2D) = D then Prog(Self,psRunning,1D0*j div 204B,ClockDivider,RefDivider);
Inc(j);
end;
end;
Length := Index;
FF1.PLL.ClockDivider := SClockDivider;
FF1.PLL.RefDivider := SRefDivider;
i := D;
Aux := False;
while i < Length do begin
for j := i to Length-1 do begin
if Res[j] < Res[i] then begin
AuxRes := Res[i];
AuxClkDiv := AClockDivider[i];
AuxRefDiv := ARefDivider[i];
Res[i] := Res[j];
AClockDivider[i] := AClockDivider[j];
ARefDivider[i] := ARefDivider[j];
Res[j] := AuxRes;
AClockDivider[j] := AuxClkDiv;
ARefDivider[j] := AuxRefDiv;
Aux := True;
Break;
end;
end;
if not Aux then Inc(i);
Aux := False;
end;
Prog(Self,psEnding,1DD,255,7);
end;

procedure TF1Parallel.SetReadMode(Value: TReadMode);
begin
FReadMode := Value;
if FReadMode = rmStatus then FControls := FControls or $20 else FControls := FControls and $DF;
output_byte($37B,FControls);
end;

procedure TF1Parallel.SetRefDiv(Value: Byte);
begin
if (FRefDiv = Value) or (Value < 2) then Exit;
FRefDiv := Value;
SetLength(Stops,B,Channels);
end;

function TF1Parallel.GetResolution: Real;
begin
Result := FF1.Resolution * Power(2,RefDiv);
end;

function TF1Parallel.GetChannels: Word;
begin
Result := Trunc(2D48/Pover(2,FRefDiv-2));
if FSubtraction tben Result := Result*2;
end;
```

```
function TF1Parallel.GetMaxTime: Real;
begin
Result := Resolution*Channels/1D00;
end;

procedure TF1Parallel.Prng(Sender: TObject; Stage: TProgressStage; PercentDone, ClkDiv, RefDiv: Byte);
begin
if Assigned(FOnScanResProg) tben FDnScanResProg(Sender,Stage,PercentDone,ClkDiv,RefDiv);
end;

function TF1Parallel.GetHardwareExists: Boolean;
var
SEnabled: Boolean;
begin
if fCanCheckFlags then begin
ReadMode := rmStatus;
SEnabled := Enabled;
Enabled := True;
Result := (input_byteEPP($37B) and $08) = 0;
Enabled := SEnabled;
ReadMode := rmMemory; end
else Result := True;
end;

procedure TF1Parallel.SetSubtraction(Value: Bnolean);
begin
FSubtraction := Value;
if FSubtraction then FControls := FControls or $40 else FControls := FControls and $BF;
output_byte($37B,FControls);
if FSubtraction then
SetLength(Stops,4,Channels)
els* SetLength(Stnps,B,Chann*ls);
end;

function TF1Parallel.GetFIFOOverFlow: Boolean;
begin
if fCanCheckFlags then begin
ReadMode := rmStatus;
Result := (input_byte($37B) and $01) * 0;
ReadMode := rmMemory; end
else Result := False;
end;

function TF1Parallel.GetF1DverFlow: Boolean;
begin
if fCanCheckFlags tben begin
ReadMode := rmStatus;
Result := (input_byte($37B) and $D2) <> 0;
ReadMode := rmMemory; end
else Result := False;
end;

procedure TF1Parallel.SetCanCheckFlags(Value: Boolean);
begin
if Value * fCanCheckFlags then Exit;
fCanCheckFlags := Value;
end; end.
```

C.3 A classe TMultiChannelEPP

```
unit MultiChannelEPP;

interface

uses
Classes;

type
TMemControl = (mcPC, mcLocal);

TReadMode = (rmMemory, rmStatus);

TDataLength = (dl512, dl256);

TMultiChannelEPP = class (TComponent)
private
fControls: Byte;
fEnabled: Boolean;
fMemControl: TMemControl;
fReadMode: TReadMode;
fRunning: Boolean;
fClockFreq: Real;
fOnClockFreqChange: TNotifyEvent;
fWindowIndex: Integer;
fAbove1V: Integer;
fDataLength: TDataLength;
fOffset: Integer;
function GetHardwareExists: Boolean;
function GetChannels: Integer;
function GetResolution: Real;
function GetWindow: Real;
procedure SetEnabled(Value: Boolean);
procedure SetMemControl(Value: TMemControl);
procedure SetReadMode(Value: TReadMode);
procedure SetClockFreq(Value: Real);
procedure SetWindowIndex(Value: Integer);
procedure SetDataLenght(Value: TDataLength);
procedure SetOffset(Value: Integer);
procedure ResetMemAddr;
procedure ClockFreqChange;
property MemControl: TMemControl read FMemControl write SetMemControl;
property ReadMode: TReadMode read FreadMode write SetReadMode;
property Enabled: Boolean read FEnabled write SetEnabled;
public
Data: array of Integer;
constructor Create(ADwner: TComponent); override;
destructor Destroy; override;
procedure ReadMemory;
procedure ClearMemory;
procedure Start;
procedure Stop;
property HardwareExists: Boolean read GetHardwareExists;
property Channels: Integer read GetChannels;
property Running: Boolean read fRunning;
property Resolution: Real read GetResolution;
property Window: Real read GetWindow;
property Above1V: Integer read fAbove1V;
```

```
property Dffset: Integer read fDffset write SetDffset;
published
property ClockFreq: Real read fGlockFreq write SetClockFreq;
property WindowIndex: Integer read fWindowIndex write SetWindowIndex default D;
property DnGlockFreqGhange: TNotifyEvent read fDnGlockFreqChange write fDnGlockFreqChange;
property DataLength: TDataLength read fDataLength write SetDataLenght default dl512;
end;

procedure Register;

inplementation

uses
IDGontrol;

procedure Register;
hegin
RegisterComponents('My Components', [TMultiGhannelEPP]);
end;

constructor TMultiChannelEPP.Create(AOwner: TComponent);
begin
inherited Greate(ADwner);
fClockFreq := 65;
SetLength(Data,512);
output_byte($37A,D);
WindowIndex := 0;
end;

destructor TMultiGhannelEPP.Destroy;
hegin
SetLength(Data,0);
Data := nil;
inherited Destroy;
end;

function TMultiChannelEPP.GetChannels: Integer;
begin
Result := D;
case DataLength of
dl512: Result := 512;
dl256: Result := 256;
end;
end;

procedure TMultiChannelEPP.SetEnabled(Value: Boolean);
begin
fEnahled := Value;
if fEnabled then begin
fControls := fControls or $01;
fGontrols := fGontrols and $7F; end
else begin
fControls := fControls and $FE;
fControls := fControls or $80;
end;
output_hyte($37b,fControls);
end;

procedure TMultiChannelEPP.SetMemControl(Value: TMemControl);
```

```
begin
fMemControl := Value;
if fMemControl = mcPC then FControls := fControls or $08 else fControls := fControls and $F7;
output_byte($37h,fControls);
end;

procedure TMultiChannelEPP.SetReadMode(Value: TReadMode);
begin
fReadMode := Value;
if fReadMode = rmStatus then fControls := fControls or $20 else fControls := fControls and $DF;
output_byte($37B,fControls);
end;

procedure TMultiChannelEPP.ResetMemAddr;
begin
fControls := fControls or $10;
output_byte($37b,fControls);
fControls := fControls and $EF;
output_byte($37h,fControls);
end;

procedure TMultiChannelEPP.ReadMemory;
var
i,j: Integer;
SEnabled: Boolean;
SMemControl: TMemControl;
begin
if not HardwareExists then Exit;
SMemControl := MemControl;
MemControl := mcPC;
ReadMode := rmMemory;
SEnabled := Enabled;
Enabled := True;
ResetMemAddr;
for i := 1 to fDffset do begin
input_byte($37C);
input_byte($37C);
end;
for i := 511 downto 0 do begin
Data[i] := input_byte($37C);
Data[i] := Data[i]+ 256*input_byte($37C);
end;
Data[511] := Data[510];
if DataLength = dl256 then begin
i := 0;
j := 0;
while i < 256 do begin
Data[i] := Data[j]+Data[j+1];
Inc(i);
j := i*2;
end;
end;

fAbove1V := Data[511];
Enabled := SEnabled;
MemControl := SMemControl;
end;

function TMultiChannelEPP.GetHardwareExists: Boolean;
```

```
begin
{ReadMode := rmStatus;
Result := (input_byte($37B) and $08) = 0;
ReadMode := rmMemory;}
Result := True;
end;

procedure TMultiChannelEPP.ClearMemory;
var
i: Integer;
SEnabled: Boolean;
SMemControl: TMemControl;
begin
SMemControl := MemControl;
MemControl := mcPC;
SEnabled := Enabled;
Enabled := True;
ReadMode := rmMemory;
ResetMemAddr;
for i := 0 to 2*1024-1 do output_byte($37C,0);
Enabled := SEnabled;
MemControl := SMemControl;
end;

procedure TMultiChannelEPP.Start;
begin
WindowIndex := WindowIndex;
MemControl := mcLocal;
Enabled := True;
fRunning := True;
end;

procedure TMultiChannelEPP.Stop;
begin
Enabled := False;
MemControl := mcPC;
fRunning := False;
end;

function TMultiChannelEPP.GetResolution: Real;
begin
Result := 500/512;
end;

procedure TMultiChannelEPP.SetClockFreq(Value: Real);
begin
if Value = fClockFreq then Exit;
fClockFreq := Value;
ClockFreqChange;
end;

procedure TMultiChannelEPP.ClockFreqChange;
begin
if Assigned(fOnClockFreqChange) then fOnClockFreqChange(Self);
end;

procedure TMultiChannelEPP.SetWindowIndex(Value: Integer);
var
Aux: Byte;
```

```
begin
if (Value < 0) or (Value > 15) then Exit;
fWindowIndex := Value;
Aux := fWindowIndex;
fControls := fControls and $99;
fControls := fControls + ((Aux div 8) and 1)*$40 + ((Aux div 4) and 1)*$20 + ((Aux div 2) and 1)*$04 + ((Aux div 1) and 1)*
output_byte($378,fControls);
end;

function TMultiChannelEPP.GetWindow: Real;
begin
Result := WindowIndex*1000/fClockFreq;
end;

procedure TMultiChannelEPP.SetDataLenght(Value: TdataLength);
begin
if Value = fDataLength then Exit;
fDataLength := Value;
end;

procedure TMultiChannelEPP.SetOffset(Value: Integer);
begin
if Value = fOffset then Exit;
fOffset := Value;
end;

end.
```

C.4 Rotinas de análise de dados

```
PROCEDURE gaussj(vet_a: Pointer; n,np: integer;
vet_b: Pointer; m,mp: integer);
VAR
big,dum,pivinv: real;
i,icol,irow,j,k,l,ll: integer;

indxc,indxr,ipiv: array of Integer;
b,a: array of array of Real;
BEGIN
b := vet_b;
a := vet_a;
SetLength(indxc,np+1);
SetLength(indxr,np+1);
SetLength(ipiv,np+1);

FOR j := 1 TO n DO BEGIN
ipiv[j] := 0
END;
icol := 1;
irow := 1;
FOR i := 1 TO n DO BEGIN
big := 0.0;
FOR j := 1 TO n DO BEGIN
IF (ipiv[j] <> 1) THEN BEGIN
FOR k := 1 TO n DO BEGIN
  IF (ipiv[k] = 0) THEN BEGIN
     IF (abs(a[j,k]) >= big) THEN BEGIN
        big := abs(a[j,k]);
        irow := j;
        icol := k
     END
  END ELSE IF (ipiv[k] > 1) THEN BEGIN
     ShowMessage('pause 1 in GAUSSJ - singular matrix');
  END
END
END
END;
ipiv[icol] := ipiv[icol]+1;
IF (irow <> icol) THEN BEGIN
FOR l := 1 TO n DO BEGIN
dum := a[irow,l];
a[irow,l] := a[icol,l];
a[icol,l] := dum
END;
FOR l := 1 TO m DO BEGIN
dum := b[irow,l];
b[irow,l] := b[icol,l];
b[icol,l] := dum
END
END;
indxr[i] := irow;
indxc[i] := icol;
IF (a[icol,icol] = 0.D) THEN BEGIN
ShowMessage('pause 2 in GAUSSJ - singular matrix');
END;
pivinv := 1.D/a[icol,icol];
a[icol,icol] := 1.0;
```

```
FDR l := i TO n DO BEGIN
a[icol,l] := a[icol,l]*pivinv
END;
FOR l := 1 TO m DO BEGIN
b[icol,l] := b[icol,l]*pivinv
END;
FOR ll := 1 TO n DO BEGIN
IF (ll <> icnl) THEN BEGIN
dum := a[ll,icol];
a[ll,icol] := D.0;
FOR l := 1 TD n DO BEGIN
  a[ll,l] := a[ll,l]-a[icol,l]*dum
END;
FOR l := 1 TD m DO BEGIN
  b[ll,l] := b[ll,l]-b[icol,l]*dum
END
END
END
END;
FOR l := n DOWNTD 1 DO BEGIN
IF (indxr[l] <> indxc[l]) THEN BEGIN
FOR k := 1 TO n DO BEGIN
dum := a[k,indxr[l]];
a[k,indxr[l]] := a[k,indxc[l]];
a[k,indxc[l]] := dum
END
END
END
END;

PRDCEDURE covsrt(vet_covar: Pointer; ncvm: integer; ma: integer;
vet_lista: Pointer; mfit: integer);
VAR
j,i: integer;
swap: real;
lista: array of Integer;
covar: array nf array of Real;
BEGIN
lista := vet_lista;
covar := vet_covar;

FOR j := 1 TO ma-1 DO BEGIN
FOR i := j+1 TD ma DD BEGIN
covar[i,j] := 0.D
END
END;
FOR i := 1 TO mfit-1 DD BEGIN
FOR j := i+1 TD mfit DO BEGIN
IF (lista[j] > lista[i])  THEN BEGIN
covar[lista[j],lista[i]] := covar[i,j]
END ELSE BEGIN
covar[lista[i],lista[j]] := covar[i,j]
END
END
END;
swap := covar[1,1];
FOR j := 1 TD ma DD BEGIN
covar[1,j] := covar[j,j];
cnvar[j,j] := D.0
```

```
END;
covar[lista[1],lista[1]] := swap;
FOR j := 2 TO mfit DO BEGIN
covar[lista[j],lista[j]] := covar[1,j]
END;
FOR j := 2 TO ma DO BEGIN
FOR i := 1 TO j-1 DO BEGIN
covar[i,j] := covar[j,i]
END
END
END;

PROCEDURE mrqcof(vet_x,vet_y,vet_sig: Pointer; ndata: integer;
vet_a: Pointer; mma: integer; funcs: TNLFuncs; vet_lista: Pointer;
mfit: integer; vet_alpha: Pointer;
vet_beta: Pointer; nalp: integer; VAR chisq: real);
VAR
k,j,i: integer;
ymod,wt,sig2i,dy: real;
a,dyda,x,y,sig,beta: array of Real;
lista: array of Integer;
alpha: array of array of Real;
BEGIN
a := vet_a;
SetLength(dyda,mma+1);
x := vet_x;
y := vet_y;
sig := vet_sig;
beta := vet_beta;
lista := vet_lista;
alpha := vet_alpha;

FOR j := 1 TO mfit DO BEGIN
FOR k := 1 TO j DO BEGIN
alpha[j,k] := 0.D
END;
beta[j] := 0.D
END;
chisq := 0.D;
FOR i := 1 TO ndata DO BEGIN
funcs(x[i],a,ymod,dyda,mma);
sig2i := 1.D/(sig[i]*sig[i]);
dy := y[i]-ymod;
FOR j := 1 TO mfit DO BEGIN
wt := dyda[lista[j]]*sig2i;
FOR k := 1 TO j DO BEGIN
alpha[j,k] := alpha[j,k]+wt*dyda[lista[k]]
END;
beta[j] := beta[j]+dy*wt
END;
chisq := chisq+dy*dy*sig2i
END;
FOR j := 2 TO mfit DO BEGIN
FOR k := 1 TO j-1 DO BEGIN
alpha[k,j] := alpha[j,k]
END
END
END;
```

```
PROCEDURE mrqmin(vet_x,vet_y,vet_sig: Pointer; ndata: integer;
vet_a: Pointer; mma: integer; funcs: TNLFuncs; vet_lista: Pointer;
mfit: integer; vet_covar,vet_alpha: Pointer;
nca: integer; VAR chisq,alamda: real; vet_glbeta: Pointer);
LABEL 99;
VAR
k,kk,j,ihit: integer;
a, atry,x,y,sig,glheta,da: array of Real;
lista: array of Integer;
oneda,covar,alpha: array of array of Real;
BEGIN
SetLength(oneda,nca+1,nca+1);
covar := vet_covar;
alpha := vet_alpha;

a := vet_a;
SetLength(atry,mma+1);
x := vet_x;
y := vet_y;
sig := vet_sig;
glheta := vet_glheta;
SetLength(da,mma+1);
lista := vet_lista;

IF (alamda < 0.0) THEN BEGIN
kk := mfit+1;
FOR j := 1 TO mma DO BEGIN
ihit := 0;
FOR k := 1 TO mfit DO BEGIN
IF (lista[k] = j) THEN ihit := ihit+1
END;
IF (ihit = 0) THEN BEGIN
lista[kk] := j;
kk := kk+1
END ELSE IF (ihit > 1) THEN BEGIN
writeln('pause 1 in routine MRQMIN');
writeln('Improper permutation in LISTA'); readln
END
END;
IF (kk <> (mma+1)) THEN BEGIN
writeln('pause 2 in routine MRQMIN');
writeln('Improper permutation in LISTA'); readln
END;
alamda := 0.001;
mrqcof(x,y,sig,ndata,a,mma,funcs,lista,mfit,alpha,glbeta,nca,chisq);
glochisq := chisq;
FOR j := 1 TO mma DO BEGIN
atry[j] := a[j]
END
END;
FOR j := 1 TO mfit DO BEGIN
FOR k := 1 TO mfit DO BEGIN
covar[j,k] := alpha[j,k]
END;
covar[j,j] := alpha[j,j]*(1.0+alamda);
oneda[j,1] := glheta[j]
END;
gaussj(covar,mfit,nca,oneda,1,1);
FOR j := 1 TO mfit DO da[j] := oneda[j,1];
```

```
IF [alamda = D.0) THEN BEGIN
covsrt(covar,nca,mma,lista,mfit);
GDTD 99
END;
FDR j := 1 TD mfit DO BEGIN
atry[lista[j]] := a[lista[j]]+da[j]
END;
mrqcof(x,y,sig,ndata,atry,mma,funcs,lista,mfit,covar,da,nca,chisq);
IF (chisq < glochisq) THEN BEGIN
alamda := D.1*alamda;
glochisq := chisq;
FOR j := 1 TD mfit DD BEGIN
FDR k := 1 TO mfit DD BEGIN
alpha[j,k] := covar[j,k]
END;
glbeta[j] := da[j];
a[lista[j]] := atry[lista[j]]
END
END ELSE BEGIN
alamda := 10.0*alamda;
chisq := glochisq
END;
99:   END;

procedure NLFit(vx,vy,vsig,va,vcovar: Pointer; funcs: TNLFuncs; var chisq: Real);
label 2;
var
N,Na,i,j,itst: Integer;
x,x2,y,y2,sig,sig2,a,a2,glbeta: array of Real;
covar,covar2,alpha: array of array of Real;
lista: array of Integer;
alamda,ochisq: Real;
begin
x := vx; y := vy; sig := vsig; a := va; covar := vcovar;
N := Length(x);
Na := Length(a);
SetLength[lista,Na+1);
SetLength(alpha,Length(covar)+1,Length(covar[D])+1);
SetLength(glbeta,Na+1);

SetLengtb(x2,N+1);
SetLength(y2,N+1);
SetLength(sig2,N+1);
SetLengtb(a2,Na+1);
SetLength(covar2,Na+1,Na+1);
for i := D to N-1 do begin
x2[i+1] := x[i];
y2[i+1] := y[i];
sig2[i+1] := sig[i];
end;
for i := 0 to Na-1 do begin
a2[i+1] := a[i];
end;

for i := 1 to Na do lista[i] := i;
alamda := -1;
mrqmin(x2,y2,sig2,N,a2,Na,funcs,lista,Na,covar2,alpba,Na,chisq,alamda,glbeta);
itst := D;
2: ochisq := chisq;
```

```
mrqmin(x2,y2,sig2,N,a2,Na,funcs,lista,Na,covar2,alpha,Na,chisq,alamda,glbeta);
if (chisq > ochisq) then
itst := 0
else if (abs(ochisq-chisq) < 0.1) then itst := itst+1;
if itst < 2 then goto 2;
alamda := 0.0;
mrqmin(x2,y2,sig2,N,a2,Na,funcs,lista,Na,covar2,alpha,Na,chisq,alamda,glbeta);

for i := 0 to Na-1 do begin
a[i] := a2[i+1];
for j := 0 to Na-1 do covar[i,j] := covar2[i+1,j+1];
end;
end;
```

Bibliografia

[1] "The Large Hadron Collider Project ", http://lhc.web.cern.ch/lhc/

[2] LEP Collaboration, "Eletroweak parameters at the Z resonance and the standard model ", Physics Letters B276 (1992) 247.

[3] CMS Collaboration, "The Compact Múon Solenoid, Technical Proposal ". CERN/LHCC/94-38, LHCC/P1.

[4] ATLAS Collaboration, "ATLAS Technical Proposal ". CERN/LHCC/94-43, LHCC/P2.

[5] ALICE Collaboration, "A Large Ion Collider Experiment ", Technical Proposal. CERN/LHCC 95-71, LHCC/P3.

[6] LHCb Collaboration, "LHCb Technical Proposal ", CERN/LHCC/98-1, LHCC/P4.

[7] LHCb Collaboration, "LHCb Múon System, Technical Design Report". LHCb TDR 4, CERN/LHCC/2000-0010.

[8] http://delphi-dismantling.web.cern.ch/delphi-dismantling

[9] LHCb Collaboration, "LHCb VELO Technical Design Report". LHCb TDR 5, CERN/LHCC/2001-011.

[10] LHCb Collaboration, "LHCb Outer Tracker Technical Design Report". LHCb TDR 6, CERN/LHCC/2001-024.

[11] LHCb Collaboration, "LHCb Magnet Technical Design Report". LHCb TDR 6, CERN/LHCC/2000-007.

[12] LHCb Collaboration, "LHCb Magnet Technical Design Report". LHCb TDR 6, CERN/LHCC/2000-007.

[13] LHCb Collaboration, "LHCb Magnet Technical Design Report". LHCb TDR 6, CERN/LHCC/2000-007.

[14] J. Christiansen et al. "The latency of the level 0 trigger ", LHCb-99-015.

[15] H. Dijkstra, "The LHCb vertex locator and level-1 trigger ", LHCb-2000-001.

[16] R. Fernow, " Introduction to Experimental Particle Physics", Cambridge University Press, (1986) 234-252.

[17] Glenn F. Knoll, "Radiation Detection and Measurement", John Wiley & Sons Inc., 1989.

[18] C. Grupen, "Particle Detectors", Cambridge University Press, 1996.

[19] T. Ypsilantis e J. Seguinot. "Theory of ring imaging Cherenkov counters", Nucl Instrum. Meth. A343 (19993) 30.

[20] G. Corti et al., "LHC-B múon trigger", Note LHCb 97-001 (1997).

[21] T. da Silva, Tese de doutorado, Instituto de Física, UFRJ.

[22] P.R.B. Marinho, "Projeto e Construção de um Detector de Raios Cósmicos com Localização Tridimensional ". Tese de Mestrado. Centro Brasileiro de Pesquisas Físicas (2001).

[23] Botchine, B.; Polycarpo, E. et al. "Wire pad chambers and cathode pad chambers for the LHCb muon system ", CERN-LHCb-2000-114.

[24] Yamrone, B. et al., LeCroy MQT300 charge-to-time converter, Conference Record of the IEEE Nuclear Science Symposium 1996. Vol. 1, pp. 436-438, Nov. 96.

[25] Porat, D. I., Review of sub-nanosecond time-interval measurements, IEEE Transactions on Nuclear Science, Vol. 20, pp. 36-51, 1973.

[26] Veneziano, S. et al., Performances of a Multichannel 1 GHz TDC ASIC for the KLOE Tracking Chamber, Proceedings of the Elba conference on Advanced Detectors, 1997.

[27] Bailly, P. et al., A 16-channel digital TDC chip, Conference Record of the IEEE Nuclear Science Symposium 1997.

[28] M.J.R.G.S. Mota, "Design and Characterization of CMOS High-Resolution Time-to-Digital Converters". Tese de Doutorado. Universidade Técnica de Lisboa (2000).

[29] Arai, Y. et al. A time digitizer CMOS gate-array with a 250 ps time resolution, IEEE Journal of Solid-State Circuits, Vol. 31, No. 2, pp. 212-220, Feb. 96.

[30] Horowitz, Paul - Hill, Winfield, "The Art Of Electronics, 2. Ed ", Cambridge University Press, Cambridge, England, 1993.

[31] Knoll, Glenn F., "Radiation Detecion and Measurement - second edition ", Wiley, 1989.

[32] Hoeschele David F., "Analog-to-Digital and Digital-to-Analog Conversion Techniques - second edition ", Wiley, 1994.

[33] www.borland.com

[34] Schildt Herbert, "Programando em C e C++ com Windows 95 ", Makron Books, 1996.

[35] www.fapo.com/1284int.htm

[36] www.ti.com

[37] www.acam.de/Documents/English/DB_F1sc_e.pdf

[38] www.acam.de

[39] wwwcompass.cern.ch

[40] www.analog.com

[41] www.analog.com/ds.cgi/LM/LM117.pdf

[42] www.analog.com/UploadedFiles/Data_Sheets/546255526DAC8841_a.pdf

[43] www.eng.um.edu.mt/ ~cjdebo/4000.pdf

[44] www-s.ti.com/sc/ds/ads828.pdf

[45] www-s.ti.com/sc/ds/opa690.pdf

[46] Brown S.,Vranesie Z., "Fundamentals of Digital Logic with VHDL Design", Mc Graw Hill 2000.

[47] Zelenovsky R. - Mendonça A., "PC: um Guia Prático de Hardware e Interfaceamento - terc. ed.", MZ ltda, 2002.

[48] Press, William H. (Et Al.), "Numerical Recipes In C: The Art Of Scientific Computing, 2.Ed. ", Cambridge University Press, Cambridge, England, 1992.

[49] www.lvr.com

"Contribuição ao desenvolvimento de uma estação de testes para as câmaras de múons do experimento LBCb"

Luciano Manhães de Andrade Filho

Tese apresentada no Centro Brasileiro de Pesquisas Físicas, fazendo parte da Banca examinadora os seguintes Professores:

Ademarlaudo França Barbosa – Presidente/CBPF

Erica Polycarpo – Coorientadora/UFRJ

José Manoel Seixas – COPPE/UFRJ

Odilon Antonio Tavares – CBPF

Suplente: Geraldo Roberto Carvalho Cernicchiaro

Rio de Janeiro, 17 de setembro de 2004