Ministério da
Ciência e Tecnologia

INPE-14123-NTC/329

DESCRIÇÃO PARCIAL DO HARDWARE E DO SOFTWARE PARA UM RECONSTRUTOR DIGITAL DE SINAIS

Suely Silva
Paulo Giácomo Milani

Registro do documento original:
<http://urlib.net/sid.inpe.br/iris@1916/2006/01.05.18.06>

INPE
São José dos Campos
2006

PUBLICADO POR:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3945-6911/6923
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

RESUMO (ABSTRACT)

Apresenta-se um sistema concebido e realizado no INPE para a Reconstrução Digital de Sinais (Digital Signal Reconstruction) que teve o objetivo de complementar o projeto de um Gerador de Sinais GPS desenvolvido totalmente em software. O Gerador de Sinais gera dados referentes a todos os sinais que um receptor GPS receberia em sua antena em um conjunto de condições diferentes, mas é o presente reconstrutor que transforma esses dados em sinais reais que poderão ser então processados no receptor. O sinal é gerado em banda base, com 8 bits de precisão, com taxa de 8Mbytes por segundo.

SUMÁRIO

1. Introdução

Este documento apresenta a descrição de um sistema concebido para uso em um Gerador de sinais GPS (GSGPS). É composto de duas partes distintas. Um software para a simulação da constelação de satélites GPS e dos sinais gerados em cada um destes últimos, para a simulação da propagação do sinal através do espaço e da atmosfera terrestre até atingir o receptor. Essa simulação utiliza-se de técnicas de processamento digitais de sinais e de modelos de todas as perturbações sofridas pelos sinais de rádio-frequência desde a saída deles da antena do satélite até a chegada à antena do receptor. Toda essa informação é armazenada na forma de um imenso arquivo que é gerado em tempo não real.

O hardware, o principal objetivo deste documento, tem a tarefa de ler em tempo real os dados desse arquivo e de fornecê-los para uma eletrônica de conversão Digital/Analógica para então ser transformado em sinal de rádio-frequência, agora na freqüência de da banda L1 correspondente ao sinal GPS (1.575,42MHz), para que o receptor possa recebê-lo. O presente texto se concentra na tarefa de tirar os dados de uma winchester tipo SCSI em um computador tipo Dell, dupla CPU Xeon de 2GHz, e fornecê-los a uma taxa constante de 8Mbytes/s, sendo que a taxa é ditada por um padrão de referência externo ao PC.

Com a referência externa torna-se possível o uso de padrões de referência de freqüência muito mais precisos do que os disponíveis em um IBM-PC, um requisito para a geração de sinais GPS.

O sistema proposto foi testado nos computadores do Laboratório de Simulação – LabSim da DMC do INPE e chegou a taxas de transferência de dados de 10Mbytes/s, ou seja, 25% acima do necessário.

O capítulo 2 apresenta o *hardware* utilizado para a transferência dos dados. É feita uma descrição das características da placa, mostrando detalhes de configuração e os modos de operação da mesma.

O capítulo 3 faz uma discussão sobre os modos de operação mais adequados à satisfação dos requisitos colocados sobre o a transferência de dados a ser realizada, apontando as diferenças entre eles e as circunstâncias em que cada um podereria ser mais indicado.

O capítulo 4 descreve a técnica de transferência de dados através de Buffer Duplo utilizada no programa, por meio das funções do *driver* da placa, para viabilizar a transferência contínua de um grande volume de dados, como o especificado.

O capítulo 5 faz a descrição das funções NI-DAQ utilizadas no programa desenvolvido, apresentando a sintaxe de cada função, sua finalidade e uma explicação sobre os parâmetros de cada uma delas.

O capítulo 6 faz o detalhamento do software desenvolvido, apresentando sua estrutura funcional.

No capítulo 7 faz-se uma descrição dos testes aplicados para comprovação da eficiência do *software* e apresenta conclusões sobre o trabalho realizado.

2. Descrição do sistema

A Figura 1.1 apresenta o diagrama do fluxo de dados para a geração dos sinais do GSGPS. Como pode ser visto, após o processamento de todos os dados, estes são armazenados em disco em tempo não real, para posterior reconstrução digital.

A interface do sistema com o operador é feita através de um programa em Visual Basic que permite a entrada de dados de configuração do simulador. Dados podem ser obtidos via rede para atualização dos dados orbitais de cada um dos satélites GPS, caso necessário. Pode-se, também, especificar a dinâmica a que o receptor GPS estará sujeito, por exemplo, se estará a bordo de um satélite, de um automóvel ou um foguete. Depois de se detalhar todos os requisitos necessários é possível gerar o conjunto de dados que corresponde à simulação desejada.

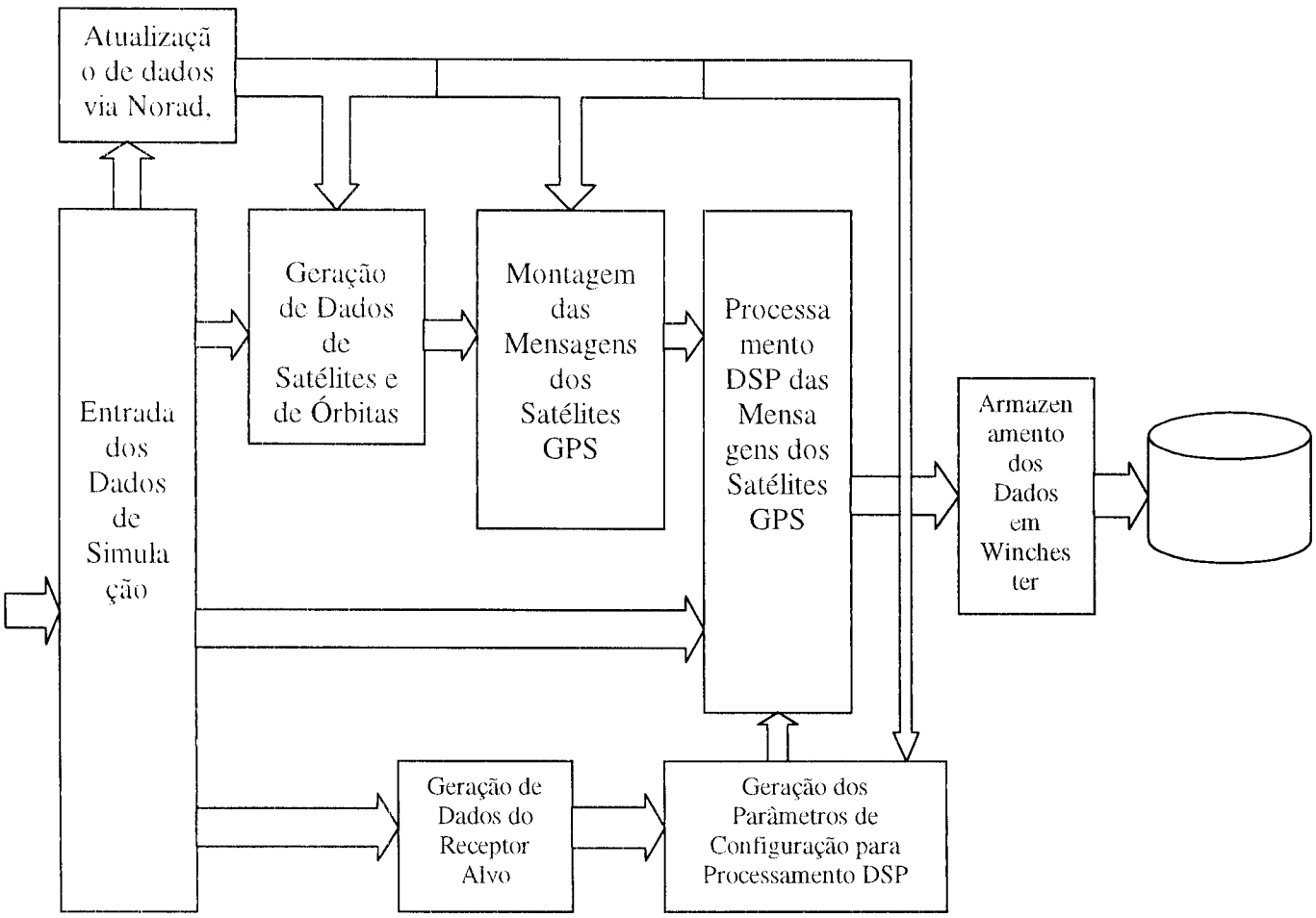

Fig. 1.1 - Fluxo de dados de processamento no simulador GPS

3. Descrição do Hardware

O hardware de transferência de dados inclui uma placa de entrada e saída (E/S) de dados de alta velocidade, e que pode ser encontrada no comércio especializado "off the shelf". Inclui também um software dedicado a controlar essa placa de E/S e que tem um driver próprio, que foi utilizado na presente proposta.

Para a realização da transferência de dados armazenados no disco rígido de um microcomputador padrão IBM-PC utilizou uma placa de E/S digital modelo PCI–6534 fabricada pela *National Instruments*. O software da aplicação em questão foi desenvolvido em linguagem C++ e utiliza rotinas pertencentes ao *driver* de dispositivo fornecido pelo fabricante para controle da placa em ambiente Windows.

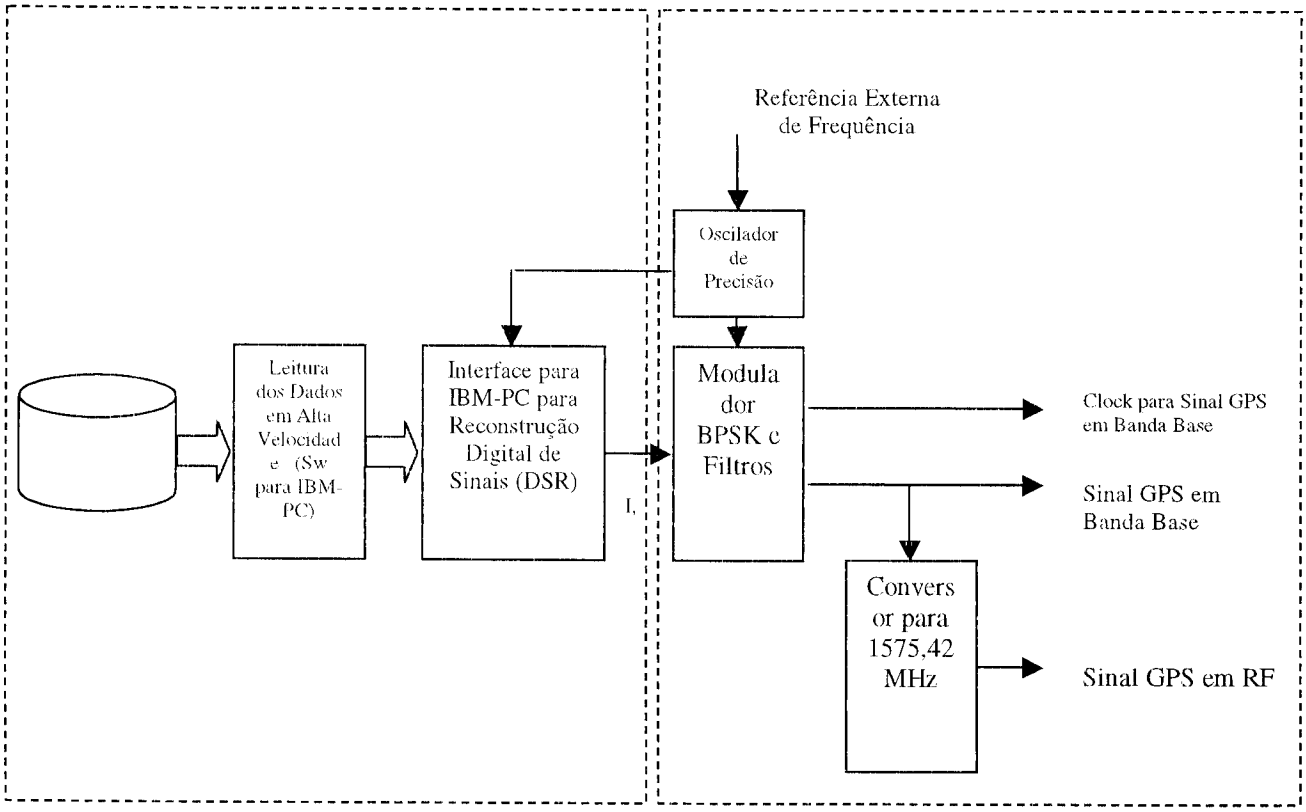

Fig. 1.2 - Diagrama de fluxo de dados para reconstrução digital e modulação

A parcela implementada no presente trabalho corresponde apenas ao quadrado da esquerda da Figura 1.2. A parte restante precisa ser projetada e implementada levando-se em conta conceitos de circuitos de RF de alta freqüência.

4. Dispositivo de Entrada/Saída - Placa NI PCI-6534

O dispositivo utilizado para a transmissão dos dados foi uma placa fabricada pela National Instruments modelo PCI 6534. Esta placa é uma interface PCI digital de entrada e saída de dados, de alta velocidade, com transmissão paralela de 32 bits. Pode ser operada com as 32 linhas individualmente configuradas, ou como portas de 8, 16, ou 32 bits. Possui dois *buffers* de memória *onboard* de 32 Mbytes cada um, que garantem taxas de transmissão de até 80 MBps. Suporta definição do estado *power-up* pelo usuário, gatilho para início e término das transferências de dados e entrada ou saída comandada por padrão ou detecção de mudança. Proporciona dois grupos de controle de temporização e *handshaking* para transferência de dados a altas velocidades. Os grupos são denominados de grupo 1 e grupo 2 e cada um possui 4 linhas de controle que podem ser usadas para temporizar as operações de entrada e saída com precisão de *hardware*. As linhas de controle são identificadas como: REQ, ACK ou STARTTRIG, STOPTRIG, e PCLK. As funções destas linhas de controle variam de acordo com o modo de operação da placa.

Os modos de operação possíveis são:

- Unstrobed I/O, aplicável para entrada ou saída digitais básicas, onde não há necessidade de *handshaking* ou *clock* provido por hardware. Aplicável também quando se deseja fazer configuração de cada bit individualmente em vez de grupos de bits;
- Unstrobed output com *driver* wired-OR, quando se quer conectar dois ou mais pinos de saída na mesma linha;
- *Handshanking* I/O com seis protocolos diferentes para comunicação com dispositivos externos utilizando troca de sinais de requisição e de reconhecimento para controle da transferência de dados.
- *Pattern* I/O para os casos em que se deve iniciar ou parar a entrada ou saída de dados de acordo com um gatilho ou transferir os dados a intervalos de tempo programados;
- *Change Detection*, para adquirir dados de entrada apenas quando determinadas linhas mudam de estado ou para monitorar atividade em determinadas linhas de entrada sem continuamente transferir dados desnecessários durante períodos de inatividade;

A interface entre o computador e a placa é realizada pelo *driver software* NI-DAQ, que vem acompanhando a placa. Ele possui uma extensa biblioteca de funções que podem ser chamadas pelo ambiente de programação da aplicação que realizará a transferência de dados. Estas funções possibilitam a utilização de todas as características da placa.

Antes da instalação da placa PCI 6534 o usuário deverá instalar o *driver software* NI-DAQ para assegurar que a placa será detectada de forma apropriada. O usuário poderá instalar o *software* seguindo as instruções do CD que o contém.

O aplicativo Mesurement & Automation Explorer (MAX), que é instalado juntamente com o NI-DAQ, fará a detecção e configuração automáticas da placa, determinando um número de identificação para a mesma. Este número será utilizado como parâmetro das funções NI-DAQ que fizerem referência a esse dispositivo.

Para instalação e configuração da placa o usuário deverá seguir as instruções contidas nas páginas 1-6 e 1-8 do manual "653X User Manual".

5. Escolha do Modo de Operação

De acordo com os requisitos, já especificados, colocados sobre a transferência de dados a ser realizada para o Simulador de Recepção de Sinais GPS, após um estudo detalhado dos modos de operação da placa, chegou-se a conclusão que os modos *Pattern Generation* e *Handshaking* com protocolo *Burst* eram adequados para atender as especificações.

O *Pattern generation*, comumente utilizado nas aplicações que necessitam de dados transferidos a uma freqüência predeterminada, não realiza *handshaking*. Um sinal TTL pode ser utilizado como gatilho para iniciar e finalizar uma operação de entrada ou saída. Esse sinal que determinará a freqüência na qual os dados serão transferidos deve ser conectado à linha REQ e pode ser gerado externamente (requisição externa) ou pela própria placa (requisição interna). Neste último caso o pulso de requisição fica presente na linha REQ como padrão. Se a linha REQ pulsa e o dispositivo periférico ou a placa não estão prontos para a transferência, então os dados são perdidos.

O *Handshaking* com protocolo *Burst* é um modo síncrono que permite transferir dados a altas velocidades e controlar a freqüência de transmissão com um sinal de pulso. Nesse modo a placa utiliza três linhas de controle na transferência de dados, os sinais REQ, ACK e PCLK. A placa 6534 sinaliza na linha ACK quando pronta para uma transferência e o dispositivo periférico sinaliza na linha REQ quando pronto. Qualquer um dos dois, a placa 6534 ou o dispositivo periférico podem gerar o sinal PCLK. A transferência ocorre na borda de subida do sinal PCLK I desde que ambas as linhas REQ e ACK estejam ativadas. Se a transferência de dados está muito rápida, tanto a placa PCI 6534 quanto o dispositivo periférico podem desabilitar as linhas ACK ou REQ respectivamente, causando uma pausa no processo de transferência. Quando qualquer dos dois dispositivos estiver pronto novamente para continuar a transferência ele habilita outra vez sua linha de controle.

Sempre que o dispositivo periférico tiver condições de monitorar o sinal ACK da placa 6534, então o modo *Handshaking-Burst* é a melhor opção de configuração da placa 6534. No entanto se o dispositivo periférico não tem meios para monitorar o sinal da linha ACK, o modo *Pattern Generation* deve então ser usado.

6. Transferência de Dados com Double Buffered

Para grande parte das aplicações de transferência de dados utiliza-se a técnica de Buffer Singular. Nessa técnica um número fixo de amostras de dados é adquirido a uma certa taxa e armazenado na memória do computador em uma operação de aquisição de dados. Posteriormente esses dados são processados ou enviados ao disco rígido para uso futuro. No caso da saída de dados, o computador envia um número fixo de amostras de um buffer na memória para um dispositivo externo a uma taxa especificada. Após a transferência o buffer é então atualizado. A implementação da transferência por Buffer Singular é relativamente simples e tem a vantagem de poder utilizar toda a capacidade de velocidade da placa. No entanto a quantidade de dados que pode ser transmitida fica limitada à quantidade de memória livre disponível no computador utilizado para a operação de entrada ou saída de dados.

Para fazer a transferência de uma grande quantidade de dados continuamente sem interrupções, faz-se necessário a utilização de técnicas de transferência mais sofisticadas. O software NI-DAQ utiliza técnicas de buffer duplo (Double-Buffering) para essa situação.

Nas operações de entrada ou saída de dados com buffer duplo, o buffer de dados é configurado como um buffer circular. Numa operação de entrada a placa começa a transferir os dados para o início do buffer até que o final seja alcançado. A placa 6534 então retorna para o início do buffer e o preenche novamente com dados. Esse processo se repete indefinidamente até ser interrompido por um erro de hardware ou pela chamada de uma função com esta finalidade. Nas operações de saída de dados a placa 6534 transfere os dados do buffer circular para a saída. Quando o final do buffer é alcançado a placa começa a transferir os dados do início do buffer, novamente.

Ao contrário das operações com buffer singular, as operações com buffer duplo reutilizam o mesmo buffer, podendo assim enviar ou adquirir um número infinito de amostras sem ter disponível uma memória infinita. A técnica do buffer duplo só será útil, porém se houver meios de acesso aos dados para atualização, armazenamento e processamento. A seguir será feita uma explicação sobre como são acessados os dados nas operações de entrada e saída.

6.1 Operações de Entrada

O buffer de dados na técnica de buffer duplo para operações de entrada é configurado como um buffer circular. Esse buffer é dividido logicamente em duas partes iguais. Essa divisão permite ao driver NI-DAQ coordenar o acesso aos dados do buffer através da placa 6534. O esquema de coordenação consiste em copiar dados do buffer circular em metades seqüenciais para um outro buffer criado, chamado buffer de transferência. Os dados do buffer de transferência podem ser processados ou armazenados conforme o necessário.

A operação de entrada inicia-se quando a placa 6534 começa a escrever dados na primeira metade do buffer circular (figura 6.1a). Depois que a Placa começa a escrever na segunda metade do buffer circular, O driver NI-DAQ pode copiar os dados da primeira metade para o buffer de transferência (figura 6.1b) e em seguida dar-lhe destino de acordo com as necessidades da aplicação. Depois que a placa encheu a segunda metade do buffer circular, ela retorna à primeira metade do buffer e subscreve os dados antigos. O driver NI-DAQ pode então copiar a segunda metade do buffer circular para o buffer de transferência (figura 6.1c). Os dados no buffer de transferência estão novamente disponíveis para utilização na aplicação.

O processo pode se repetir indefinidamente, gerando assim um fluxo contínuo de dados para a aplicação. A figura 6.1d é equivalente ao passo na figura 6.1b e é o início de outro ciclo de dois passos.

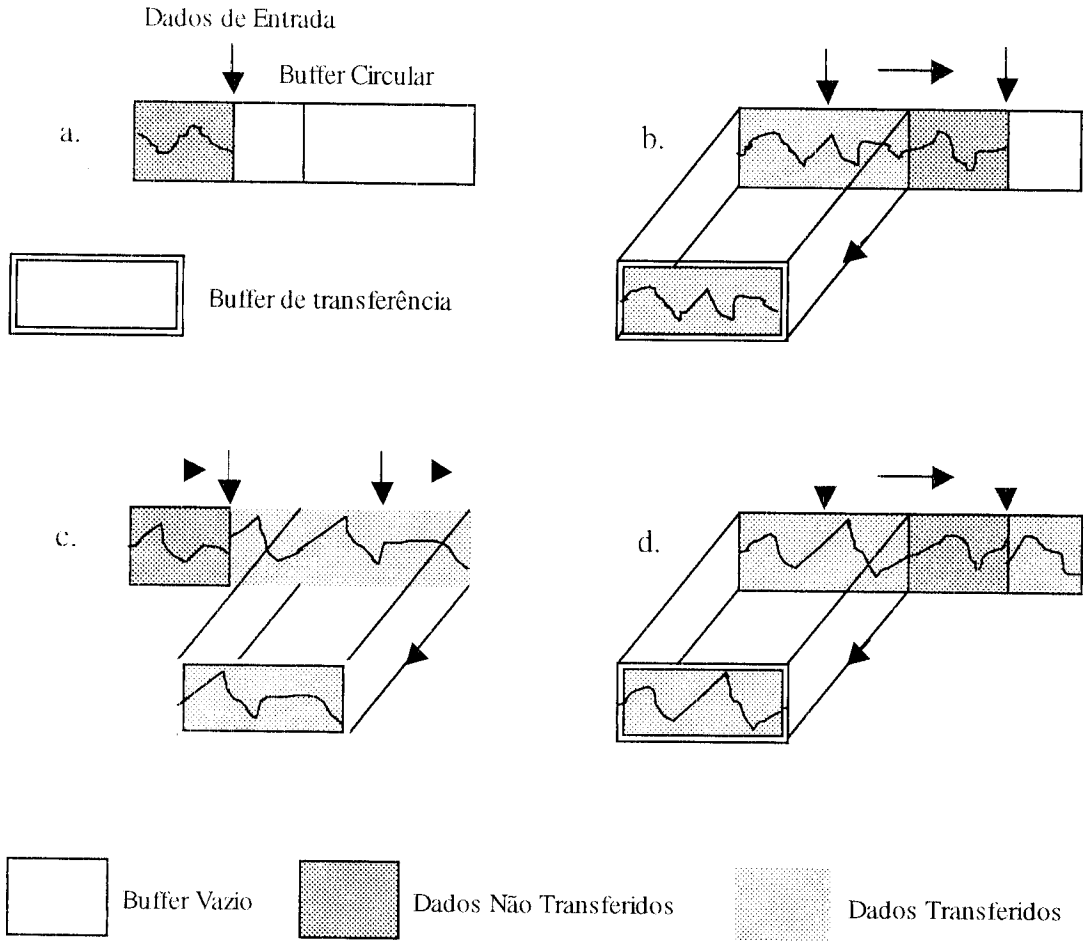

Figura 6.1 – Entrada com Buffer Duplo e Transferência de Dados Seqüenciais

Esse esquema de coordenação para entrada pode eventualmente apresentar falha. Existem dois possíveis problemas em uma aplicação para entrada de dados. O primeiro é a subscrição dos dados antes do driver NI-DAQ copiá-los para o buffer de transferência. Essa situação é ilustrada na figura 6.2.

A figura 6.2b mostra que o driver não conseguiu transferir os dados da primeira metade do buffer circular para o buffer de transferência enquanto a placa escrevia dados na segunda metade. Por essa razão a placa começa a subscrever os dados da primeira metade antes deles serem copiados para o buffer de transferência (figura 6.2c). Para garantir que os dados não serão corrompidos o driver tem que esperar que a placa termine de subscrevê-los no primeiro meio-buffer para então copiá-los para o buffer de transferência (figura 6.2d). Para essa situação o driver retorna um aviso de subscrição antes de cópia (overWriteError). Esse aviso indica que os dados no buffer de transferência são válidos, mas que houve perda de dados anteriores. As transferências subseqüentes continuam normalmente sem erro.

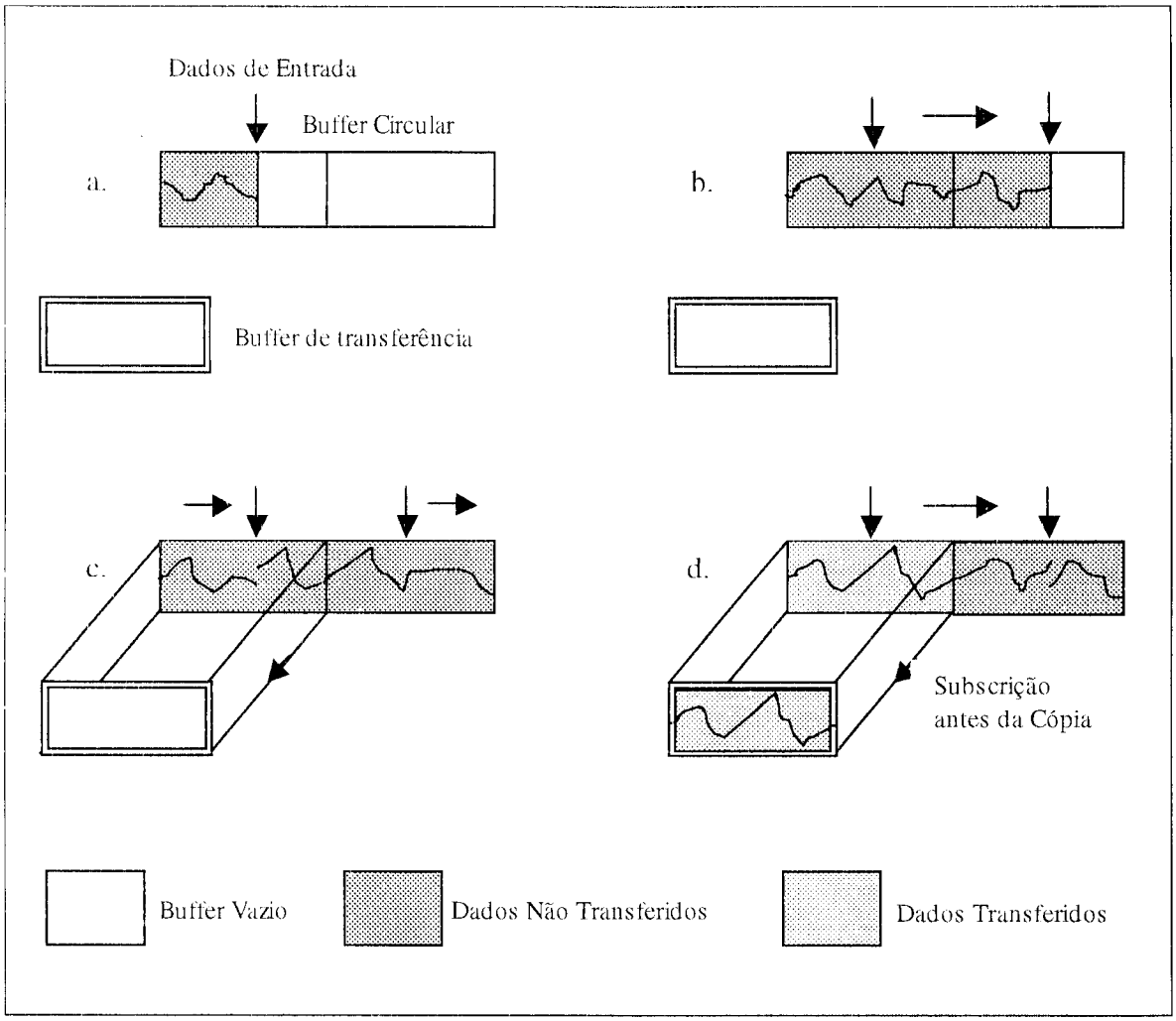

Figura 6.2 – Entrada com Buffer Duplo e Subscrição de Dados Antes de Cópia.

O segundo problema potencial ocorre quando a placa subscreve dados que o driver está simultaneamente copiando para o buffer de transferência. Novamente um erro de subscrição é retornado (overWriteError). A figura 6.3 mostra esta situação.

Na figura 6.3b, o driver começou a copiar dados do primeiro meio-buffer para o buffer de transferência. No entanto, por alguma razão ele não conseguiu copiar todo o meio-buffer antes da placa começar a subscrever os dados nesse meio-buffer (figura 6.3c). Conseqüentemente os dados copiados podem estar corrompidos. Novas transferências ocorrerão normalmente sem erro desde que não ocorram mais problemas.

6.2 Operações de Saída

As operações de saída utilizando buffer duplo são similares às operações de entrada. O buffer circular, como no caso de entrada, é dividido logicamente em duas partes. A divisão permite ao driver NI-DAQ coordenar o acesso do usuário aos dados através da placa. O driver copia os dados de um buffer de transferência criado anteriormente para o buffer circular em metades seqüenciais. Os dados do buffer de transferência são atualizados entre as transferências. A figura 6.4 ilustra a saída de dados.

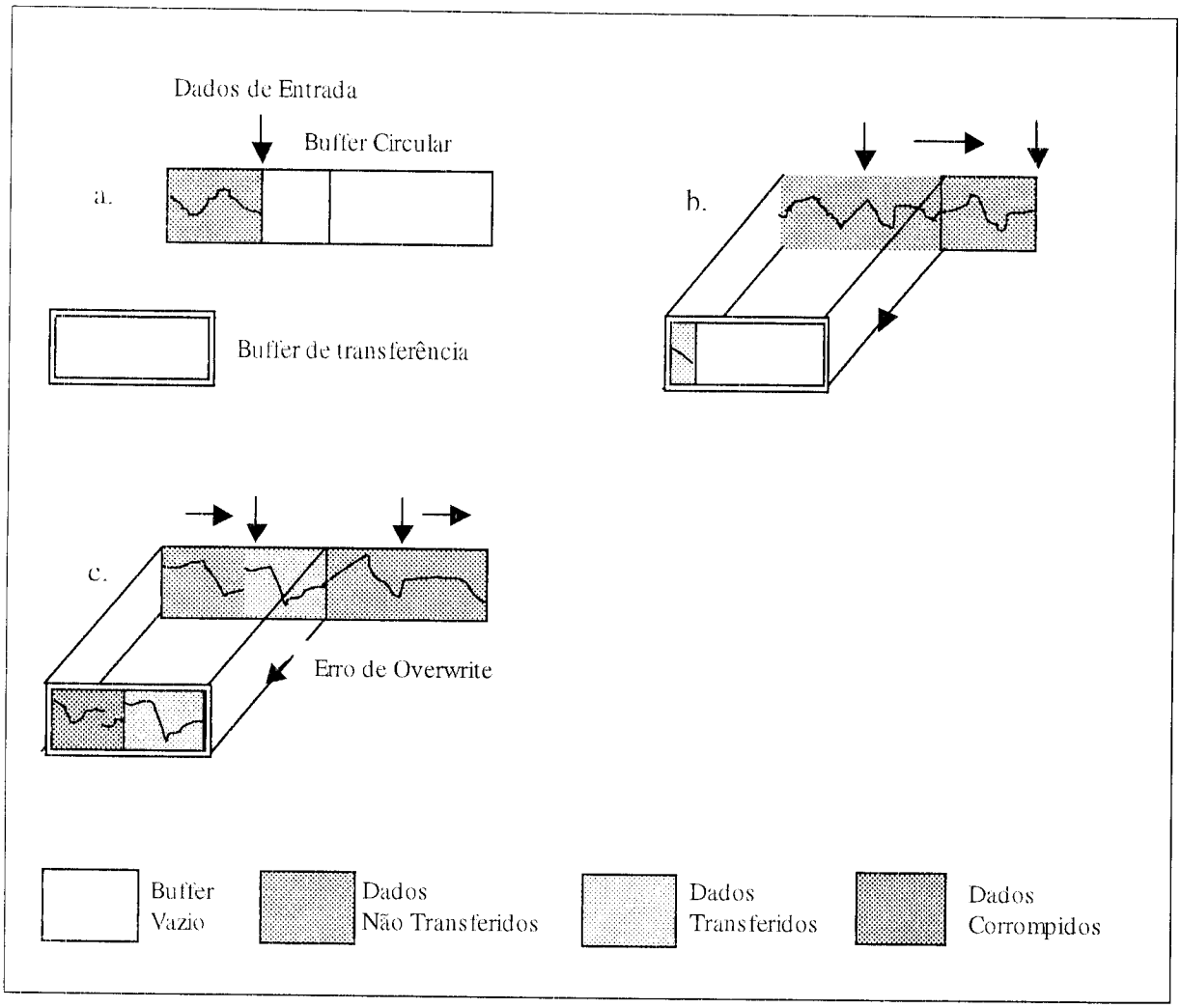

Figura 6.3 – Entrada com Buffer Duplo e Erro de Overwrite.

A operação de saída utilizando a técnica de buffer duplo começa enviando dados do primeiro meio-buffer (figura 6.4a). Depois que a placa começa a restaurar os dados do Segundo meio-buffer, o driver pode copiar os dados preparados do buffer de transferência para o primeiro meio-buffer (figura 6.4b). A aplicação pode então atualizar os dados no buffer de transferência. Depois que a placa terminou com o segundo meio-buffer, ela retorna ao primeiro meio-buffer e começa a enviar para fora os dados atualizados no primeiro meio-buffer. O driver pode agora copiar os dados do buffer de transferência para a segunda metade do buffer circular (figura 6.4c). O buffer de transferência fica então novamente disponível para atualização pela aplicação. A figura 6.4d é equivalente à figura 6.4b e é o reinício da seqüência de passos.

Como na entrada de dados, a saída utilizando a técnica de buffer duplo tem dois potenciais problemas. O primeiro é a possibilidade da placa restaurar e enviar para fora os mesmos dados antes do *driver* atualizar o buffer circular com os dados novos vindo do buffer de transferência. Esta situação é ilustrada na figura 6.5. A figura 6.5b mostra que o driver perdeu a oportunidade de copiar os dados do buffer de transferência para primeira metade do buffer circular enquanto a placa restaurava os dados da segunda metade. Como resultado, a

placa começa a enviar para fora os dados originais na primeira metade do buffer circular antes do driver tê-lo atualizado com os dados novos do buffer de transferência (figura 6.5c). Para garantir dados de saída não corrompidos, o driver é forçado a esperar até que a placa termine a restauração dos dados do primeiro meio-buffer antes de copiar os dados do buffer de transferência. Depois que a placa começar a enviar para fora a segunda metade, o driver copia os dados do buffer de transferência para o primeiro meio-buffer. Para esta situação o driver retorna um overwrite antes de um aviso de erro de cópia (overWriteError). Este aviso indica que a placa enviou dados velhos, porém não corrompidos. As transferências futuras não retornarão nenhum aviso desde que elas se mantenham sincronia com a placa como na figura 6.4.

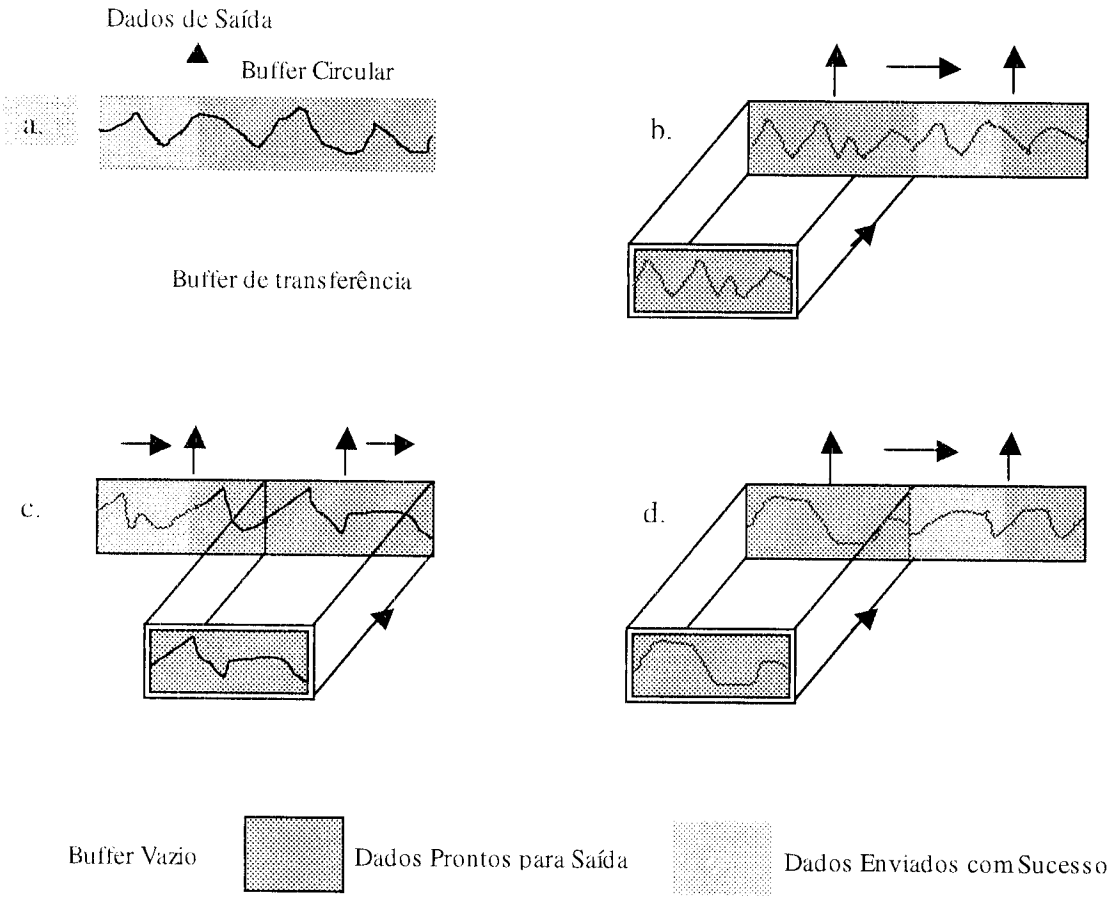

Figura 6.4 – Saída com Buffer Duplo em Transferência Seqüencial de Dados.

O segundo problema potencial acontece quando a placa restaura dados que o driver está ao mesmo tempo subscrevendo com dados do buffer de transferência. Novamente o aviso de erro (overWriteError) é retornado. A situação é apresentada na figura 6.6.

Na figura 6.6b, o driver começou a copiar dados do buffer de transferência para a primeira metade do buffer circular. Contudo, o driver não foi capaz de copiar todos os dados

antes que a placa começasse a restaurar do primeiro meio buffer (figura 6.6c). Por isso os dados de saída podem estar corrompidos, contendo dados antigos e novos. As futuras transferências ocorreram normalmente se nenhuma das condições de erro se repetirem.

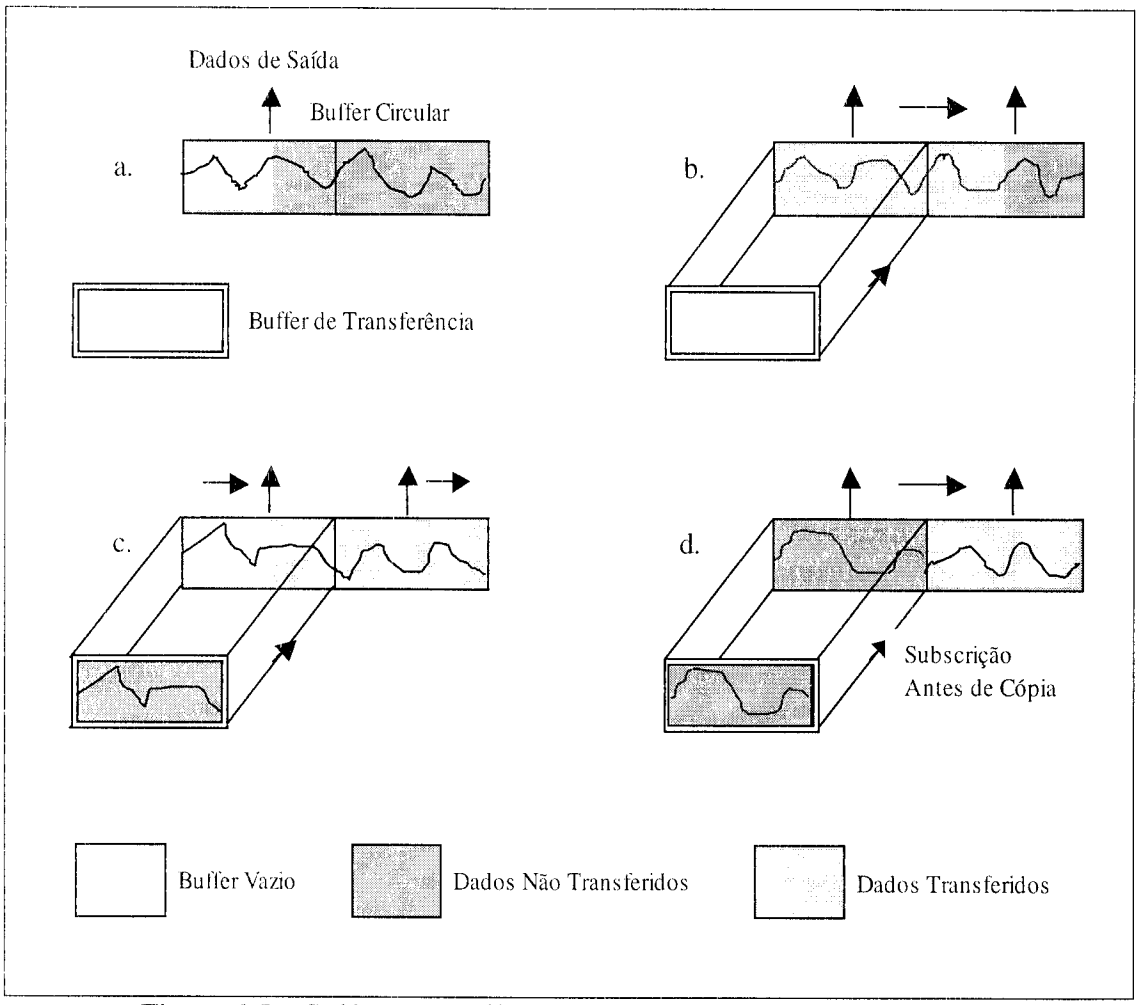

Figura 6.5 – Saída com Buffer Duplo e Subscrição Antes da Cópia.

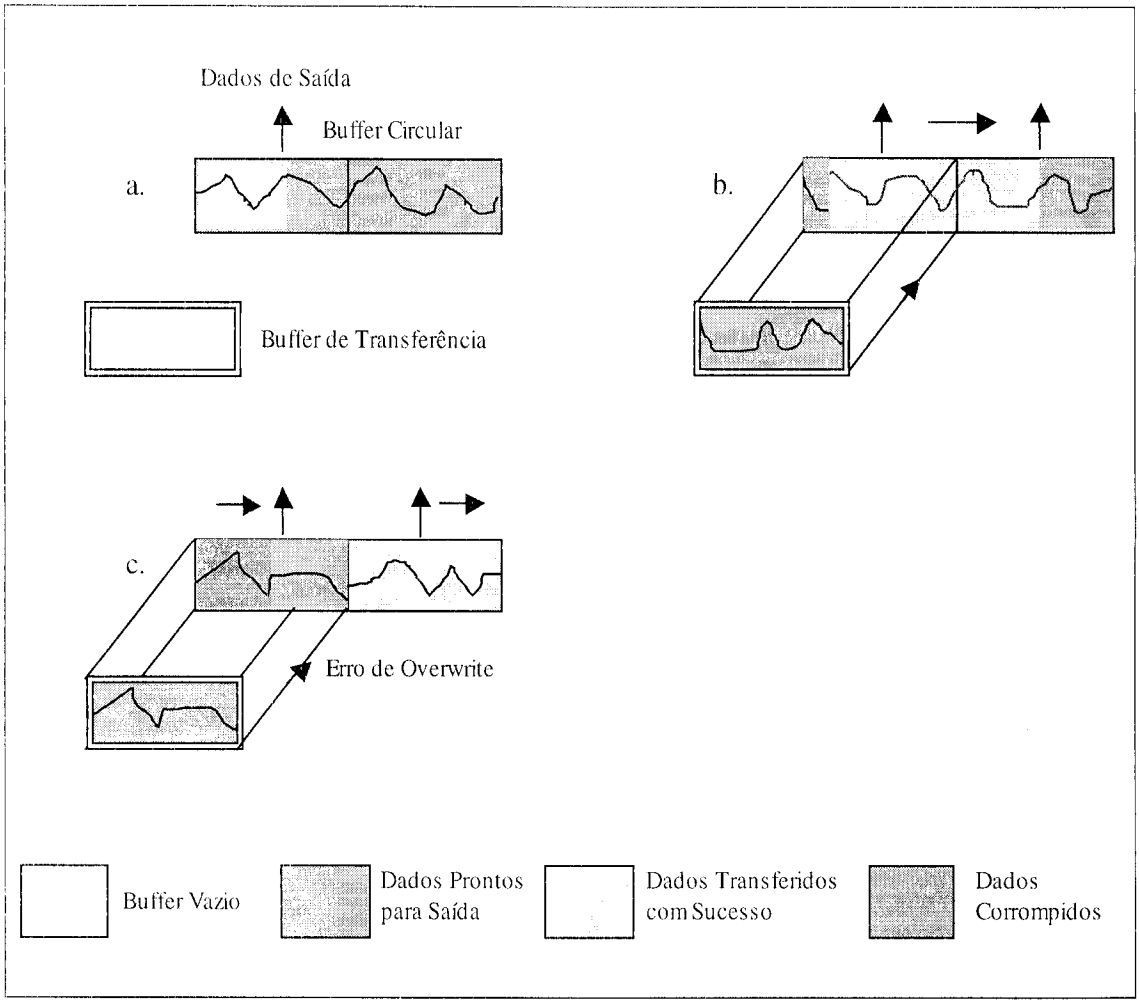

Figura 6.6 – Saída com Buffer Duplo e Erro de Subscrição.

7. Funções NI-DAQ

O conjunto de rotinas NI-DAQ realiza a interface entre o ambiente de programação utilizado pelo usuário e a placa PCI 6534, permitindo o gerenciamento de todas as suas funcionalidades. A figura 7.1, que foi adaptada do manual "653X User Manual" da National Instruments, mostra o relacionamento entre o ambiente de programação utilizado pelo usuário, o *drive software* NI-DAQ e a placa de entrada e saída de dados.

Este capítulo tem o objetivo de apresentar ao usuário a forma de utilização das funções NI-DAQ e fazer a descrição das rotinas que foram utilizadas no programa de aplicação desenvolvido.

Toda função NI-DAQ tem o seguinte formato:

status = Nome da Função (parâmetro 1, parâmetro 2 ... parâmetro n)

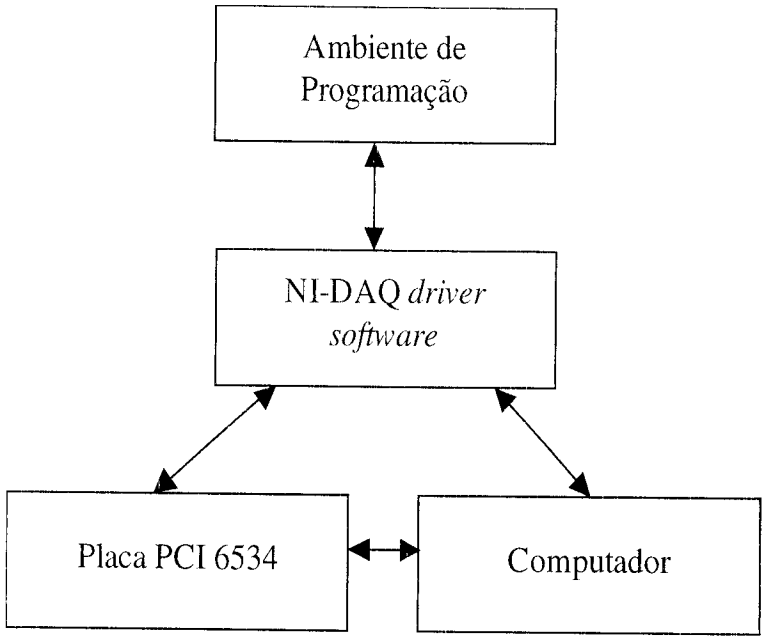

Figura 7.1 - Relacionamento entre o ambiente de programação, NI-DAQ e a Placa.

com $n \geq 0$. O valor retornado na variável **status** indica sucesso ou falha na execução da função. Este valor pode ser: negativo se a função não executar em caso de erro; zero se for executada com sucesso ou positivo quando a função for executada, porém com algum efeito potencialmente sério. Em de valor negativo para a variável **status**, o valor retornado representa um código de erro que pode ser verificado na Lista de Códigos de Status disponível na ajuda *online* "NI-DAQ Function Reference Help".

Todo o conjunto de funções NI-DAQ utililiza tipos de variáveis definidos no arquivo *header* **nidaex.h** fornecido, juntamente com o *driver*, pela National Instrumentes.

Os tipos utilizados no programa descrito neste documento são:
i16 – unsigned short
u32 – unsigned long

Descrição da Funções Utilizadas no Programa

status = Timeout_Config (deviceNumber, timeout)

Esta função estabelece um tempo limite que as funções síncronas utilizam para retornar o controle à aplicação que fez a chamada da função. Retorna um erro de timeout (**timeouterr**) se o tempo limite é excedido.

Entrada deviceNumber – identificação da placa.
Timeout –número de timer ticks. A duração de um tick é de 55 ms. O valor -1 para este parâmetro desabilita o timeout.

status = NIDAQErrorHandler(daqStatus, functionName, ignoreWarning)

Esta função tem o objetivo de mostrar uma caixa de diálogo de mensagem de erro ou código de aviso com descrição e opção para o usuário continuar. É utilizada após cada chamada de função NI-DAQ pra verificação de erro em qualquer uma delas através da variável de retorno *status*.

Entrada daqStatus – código de erro de algum erro particular que ocorreu.
functionName – nome da função associada ao código de status.
ignoreWarning – flag para habilitar ou desabilitar a messagem de aviso.

status = NIDAQMakeBuffer(buffer, count, dataType)

Cria um buffer utilizado na saída de dados.

Entrada count – número de elementos a ser transmitido.
dataType – indica o tipo de dado a ser colocado no buffer. WFM_DATA_I16 indica dados do tipo **int** 16 bits com sinal.

Saída buffer –ponteiro para o espaço de memória alocada para o buffer.

status = DIG_Grp_Config (deviceNumber, group, groupSize, port, dir)

Configura um grupo de portas como de entrada ou saída de dados.

Entrada deviceNumber – identificação da placa.
group – indentificação do grupo. Pode assumir os valores 1 ou 2.
groupSize – dimensão do grupo. O valor 2 associa duas portas (16 bits) ao grupo;
port – indica que portas serão associadas ao grupo. Para groupSize=2, port=0 associa as portas A e B da placa ao grupo
dir - indica se o grupo de portas é para saída de dados ou entrada. O valor 1 indica que o grupo de portas é para saída de dados.

status = DIG_Block_PG_Config (deviceNumber, group, config, reqSource, timebase, reqInterval, externalGate)

Esta rotina tem a função de habilitar o modo Pattern Generation e indicar a temporização utilizada na transmissão e a fonte do sinal de clock.

Entrada
deviceNumber – identificação da placa.
group – número do grupo.
config – habilita ou desabilita o modo Pattern Generation. O valor 1 para este parâmetro habilita o modo Pattern Genetration usando "request-edge latching".
reqSource – fonte dos sinais de requisição da transmissão dos dados. O valor 1 indica fonte de requisição externa.
timebase – valor da base de tempo. O valor –3 indica uma base de tempo de 50 ns.
reqInterval – número de unidades de base de tempo entre as requisições de sinais. Utilizando a base de tempo de 50 ns o valor 4 para este parâmetro configura a transmissão de dados a uma frequência de 5 MHz.
ExternalGate – habilita ou desabilita o *gating* externo.

status = DIG_DB_Config (deviceNumber, group, dbMode, oldDataStop, partialTransfer)

Habilita ou desabilita operações com buffer duplo e estabelece as opções.

Entrada
deviceNumber – identificação da placa.
Group – grupo a ser utilizado na operação.
DbMode – habilita ou desabilita o modo de buffer duplo. Para habilitação deve-se usar o valor 1.
OldDataStop - o valor 1 para este parâmetro não permite a regeneração de dados.
PartialTransfer – utilizando o valor 0 fica desabilitada a opção de transferência parcial do conteúdo do meio-buffer nas operações de buffer duplo.

status = Align_DMA_Buffer (deviceNumber, resource, buffer, count, bufferSize, alignIndex)

Alinha os dados no buffer DMA

Entrada
deviceNumber – identificação da placa.
resource –indica o grupo para o qual o buffer está sendo utilizado. O valor 11 é utilizado para group 1 e groupsize 2.

E/S
buffer – vetor de inteiros utilizado para armazenar as amostras.
count – número de amostras de dados que o buffer contém.
bufferSize – dimensão do buffer.

Saída
alignIndex - offset dentro do vetor da primeira amostra de dados.

status = DIG_Block_Out (deviceNumber, group, buffer, **count)**

Inicia a transferência de dados da memória para o grupo especificado.

Entrada deviceNumber – identificação da placa.
group – número do grupo.
buffer – vetor contendo os dados a serem transmitidos para o grupo especificado.
count - número de itens, de acordo com a definição do grupo, a ser transferido.

status = DIG_DB_HalfReady (deviceNumber, group, halfReady)

Verifica se o próximo meio-buffer está disponível durante uma operação de buffer duplo.

Entrada deviceNumber – identificação da placa.
group – número do grupo.

Saída halfReady – indica se o próximo meio-buffer de dados está pronto para transferência. Se o valor retornado for 1 então o meio-buffer está pronto para transferência através da função DIG_DB_Transfer.

status = DIG_DB_Transfer (deviceNumber, group, halfBuffer, ptsTfr)

Gerencia um tempo de espera para o driver NI-DAQ transferir os dados de um buffer de transferência para o buffer duplo de saída de dados.

Entrada deviceNumber – identificação da placa.
group – número do grupo.
PtsTfr – utilizado nos casos onde a transferência do conteúdo parcial do último meio-buffer é habilitada. No caso da não habilitação desta opção esse parâmetro é ignorado.

E/S halfBuffer – vetor que armazena os dados a serem transferidos.

status = NIDAQYield(yieldMode)

Tem a função de permitir ou não o processmento simultâneo de outros eventos no sistema.

Entrada yieldMode – qualquer valor diferente de zero assumido por este parâmetro indicará a permissão para o processamento de outors eventos no sistema.

status = DIG_Block_Clear (deviceNumber, group)

Esta função finaliza qualquer transferência assíncrona em curso.

Entrada deviceNumber - identificação da placa
group - grupo de portas definido para transferência de dados

8. Descrição do Programa para Transferência de Dados

Para a realização da transferência de dados segundo os critérios especificados para o projeto do reconstrutor de sinais foi desenvolvido um programa em linguagem C++ utilizando o ambiente de desenvolvimento Bilder C++ da Borland. O programa realiza a transferência contínua dos dados contidos em um arquivo gravado no disco rígido do microcomputador para um equipamento periférico a uma taxa constante de 10 MBps através da placa de entrada e saída digital PCI 6534.

Foram testadas duas versões para o programa desenvolvido: uma que executa a transferência utilizando o modo de operação *Pattern Generation* e a outra utilizando o modo *Handshaking* com protocolo *Burst*. Nos dois casos é utilizada a técnica de transferência com operação de *buffer* duplo e o sincronismo é feito através da recepção de sinal de clock externo. Para a saída dos dados é utilizado um grupo de duas portas digitais de oito bits cada uma. Os códigos fontes das duas versões do programa estão respectivamente nos arquivos Dodbpg16_exc.cpp e Dbburstexc.cpp. As dua versões possuem a mesma estrutura diferindo-se apenas em detalhes relacionados ao modo de operação.

A seguir são apresentados os passos executados na versão do programa que utiliza o modo Pattern Generation e a listagem do programa fonte Dodbpg16_exc.cpp.

Passos Executados pelo programa

1. Declaração das variáveis utilizadas;
2. Reserva de memória para utilização nas operações de *buffer* duplo;
3. Ajuste de tempo limite para retorno do controle ao programa principal em caso de erro na chamada das funções do *driver* NI-DAQ;
4. Construção dos *buffers* de dados;
5. Abertura do arquivo que contém os dados e carregamento inicial do *buffer* de transferência;
6. Configuração do grupo de portas e da direção da transmissão;
7. Ajuste do protocolo de transferência a ser utilizado e da taxa de transmissão;
8. Ativação do modo de operação com *buffer* duplo;
9. Comando para início da saída de dados;
10. Inicialização de uma malha para repetição contínua da operação de saída com buffer duplo;
11. Inicialização de malha para atualização do meio-buffer;
12. Finalização das malhas de atualização de buffers e saída de dados;
13. Desativação das funções de comando de saída de dados e da operação de buffer duplo;

14. Desconfiguração do grupo;

15. Desabilitação do tempo limite de retorno e liberação da memória alocada.

Listagem do Programa Fonte

```
#pragma hdrstop
#include <condefs.h>
//---------------------------------------------------------------------
#pragma argsused

/*****************************************************************************
*
*   Dodbpg16_exc.cpp
*
* Descrição:
*   Envia dados digitais continuamente para um buffer através de um grupo de
*duas portas digitais usando contagem de tempo externa
*
*
* Tipos de tarefas:
*   BUF, NCH, INTTIM, CONT, ASYNC
*
* Parâmetros-chave:
*   iGroup, iGroupSize, iDir, iDBModeON, iOldDataStop, ulCount,
*    iHalfReady, ulBufferSize, iResource, iIgnoreWarning, ulAlignIndex
*
*
* Lista de Funções NI-DAQ usadas neste exemplo:
*   Timeout_Config, NIDAQErrorHandler, NIDAQMakeBuffer,
*    DIG_Grp_Config, DIG_Block_PG_Config, DIG_DB_Config,
*    Align_DMA_Buffer, DIG_Block_Out, DIG_DB_HalfReady,
```

```
*     DIG_DB_Transfer, NIDAQYield, DIG_Block_Clear
*
*
* Informação de pinagem:
*    Os sinais digitais sairão nas portas 0 e 1. Conecte o terra de referência
* ao pino DIG GND.
*
*****************************************************************************/
/*
 * Includes:
 */
#include "nidaqex.h"
#include "stdio.h"
#include "stdlib.h"
#include "conio.h"
/*
 * Main:
 */
USELIB("..\..\Arquivos de programas\National Instruments\NI-DAQ\Lib\nidaq32b.lib");
USELIB("..\..\Arquivos de programas\National Instruments\NI-DAQ\Lib\nidex32b.lib");
//---------------------------------------------------------------------------
void main(void)
{
  /*
   * Local Variable Declarations:
   */

  FILE *arq;
  u32 nelem;
```

```
i16 iStatus = 0;
i16 iRetVal = 0;
i16 iDevice = 1;
i16 iGroup = 1;
i16 iGroupSize = 2;
i16 iPort = 0;
i16 iDir = 1;
i16 iPgConfig = 1;
i16 iReqSource = 1;
i16 iPgTB = -3;
u16 iReqInt = 4;
i16 iExtGate = 0;
i16 iDBModeON = 1;
i16 iDBModeOFF = 0;
i16 iOldDataStop = 1;
i16 iPartialTransfer = 0;
static i16 *piBuffer;
static i16 *piHalfBuffer;
u32 ulCount = 8000000;
u32 ulHCount= 4000000;
i16 iHalfReady = 0;
u32 ulPtsTfr = 4000000;
u16 iLoopCount = 0;
u16 iHalfBufsToRead = 50 ;
u32 ulBufferSize = 16000000;
i16 iResource = 11;
i16 iIgnoreWarning = 1;
u32 ulAlignIndex = 0;
i32 lTimeout = 180;
i16 iYieldON = 1;
```

```
/* alocacao de memoria para piBuffer*/
if ((piBuffer = (i16*) malloc(ulBufferSize*sizeof(i16))) == NULL)
{
  printf("Nao há memoria suficiente para alocar o primeiro buffer\n");
  getch();
  exit(1);  /* termina o programa se nao há memoria disponivel */
}

/* alocacao de memoria para piHalfBuffer*/
if ((piHalfBuffer = (i16*) malloc(ulHCount*sizeof(i16))) == NULL)
{
  printf("Nao há memoria suficiente para alocar o segundo buffer\n");
  getch();
  exit(1);  /* termina o programa se nao há memoria disponivel */
}


/* Ajuste de um tempo limite de saída (#Sec * 18ticks/Sec) tal que se houver
  algo errado, o programa não persistirá na chamada de DIG_DB_Transfer. */

iStatus = Timeout_Config(iDevice, lTimeout);

iRetVal = NIDAQErrorHandler(iStatus, "Timeout_Config",iIgnoreWarning);

/* Preparação do buffer */

iStatus = NIDAQMakeBuffer((void*)piBuffer, ulCount, WFM_DATA_I16);

if (iStatus == 0)
```

```
{
  /* Abertura do arquivo de dados */

  arq=fopen("c:/AqDad/dados/dadg16.dat","r");
  if (arq==NULL)
    fprintf(stderr,"Não é possível abrir o arquivo.\n");
  else
    fseek(arq, SEEK_SET,0);
    nelem=fread(piBuffer,2,ulCount,arq);

  if (nelem!=ulCount)
  {
      printf("Houve erro no carregamento inicial do buffer\n");
    getch();
    exit(1);
  }
  else

  /* Configuração do grupo de portas como de saída, com Pattern
  Generation. */

  iStatus = DIG_Grp_Config(iDevice, iGroup, iGroupSize, iPort,iDir);

  iRetVal = NIDAQErrorHandler(iStatus, "DIG_Grp_Config",iIgnoreWarning);


  /* Configuração do pattern generation com clock externo, timebase
    -3 e intervalo 4. */

  iStatus = DIG_Block_PG_Config(iDevice, iGroup, iPgConfig,iReqSource,iPgTB,
```

```
            iReqInt, iExtGate);

        iRetVal                                                              =
NIDAQErrorHandler(iStatus,"DIG_Block_PG_Config",iIgnoreWarning);

        /* Ativação do modo double-buffered, com proteção contra
          overwrite do meio-buffer (iOldDataStop). */

        iStatus = DIG_DB_Config(iDevice, iGroup, iDBModeON,iOldDataStop,
            iPartialTransfer);

        iRetVal = NIDAQErrorHandler(iStatus, "DIG_DB_Config",iIgnoreWarning);

        /* Alinhamento do buffer do DMA tal que não ocorra cruzamento de fronteira
        para computadores com barramento AT. */

        /* O alinhamento só é necessário na geração do buffer, desde que o
        buffer de escrita fica cheio com a chamada de DIG_DB_Transfer.
        Note que piBuffer é na verdade duas vezes maior que o necessário,
        devido a um possível alinhamento. */


        iStatus = Align_DMA_Buffer(iDevice, iResource, piBuffer,ulCount,
            ulBufferSize, &ulAlignIndex);

        iRetVal = NIDAQErrorHandler(iStatus, "Align_DMA_Buffer",iIgnoreWarning);

        /* Inicia a saída em double-buffered pattern generation. O
        'piBuffer' aqui é o buffer de aquisição circular. */
```

```
iStatus = DIG_Block_Out(iDevice, iGroup, piBuffer, ulCount);

iRetVal = NIDAQErrorHandler(iStatus, "DIG_Block_Out",iIgnoreWarning);

/* Escreve 10 meio-buffers. */

while ((iLoopCount < iHalfBufsToRead) && (iStatus == 0))
{
  iStatus = DIG_DB_HalfReady(iDevice, iGroup, &iHalfReady);
  if (iStatus >= 0)
  {
    if (iHalfReady == 1)
    {
      /* Meio-buffer de dados será colocado no 'piHalfBuffer'. */

       nelem=fread(piHalfBuffer,2,ulHCount,arq);
       if (nelem!=ulHCount)
       {
         printf("Houve erro na atualização do meio-buffer\n");
         getch();
         exit(1);
       }
       else
       iStatus = DIG_DB_Transfer(iDevice,iGroup,piHalfBuffer,ulPtsTfr);

       iRetVal = NIDAQErrorHandler(iStatus,"DIG_DB_Transfer",
                 iIgnoreWarning);

       ++iLoopCount;
```

```
            printf("iLoopCount= %d\n",iLoopCount);

        }
    }
    else
    {
        iRetVal = NIDAQErrorHandler(iStatus,"DIG_DB_HalfReady",
                iIgnoreWarning);
    }
    iRetVal = NIDAQYield(iYieldON);
}

fclose(arq);
printf("%d meio-buffers transmitidos!\n",iHalfBufsToRead);

/* Desativação das funções - não há checagem de erros. */
/* Desativa a operação de bloco. */

iStatus = DIG_Block_Clear(iDevice, iGroup);

/* Desativa o modo DB para o dispositivo. */

iStatus = DIG_DB_Config(iDevice, iGroup, iDBModeOFF,iOldDataStop,
            iPartialTransfer);

/* Desconfiguração do grupo. */

iStatus = DIG_Grp_Config(iDevice, iGroup, 0, 0, 0);
```

```
        printf(" Saída de dados digitais por double-buffered pattern generation
realizada!\n");
        printf("Ultimo iRetVal = %d \n",iRetVal);


      }
      else
      {
        printf(" O buffer não foi criado corretamente. Checar os parâmetros em
NIDAQMakeBuffer.\n");
      }
      /* Desabilita timeouts. */
      iStatus = Timeout_Config(iDevice, -1);


      /* Liberação da memória alocada  */
      free(piBuffer);free(piHalfBuffer);


      getchar();
    }
    /* Fim do programa */
```

Caso seja necessário realizar alterações neste código fonte o usuário deverá ter instalado em seu computador o *software drive* NI-DAQ, que contém as bibliotecas e os headers necessários para compilação do programa. Durante a instalação do *software* NI-DAQ o usuário deverá fazer a opção de instalar ou não as bibliotecas e *headers* apropriadas para os ambientes Microsoft Visual C++ e Builder C++ de acordo com sua necessidade.

9. Descrição dos Testes e Conclusões

Com o objetivo de testar o software desenvolvido foi montado um ambiente de teste. A placa foi instalada inicialmente em um Pentium III 800 MHz, 128 MB de RAM e HD IDE 20 GB/7200 rpm. Foram conectados às linhas apropriadas de dados e controle respectivamente um osciloscópio e um gerador de sinais para monitoração dos dados e simulação do sinal de *clock* externo. Os testes foram iniciados com uma taxa de transmissão baixa que foi sendo aumentada gradativamente. Verificou-se que diante desta configuração a velocidade máxima de transmissão alcançada foi de 2 MBps, insuficiente para satisfazer os requisitos do projeto do Simulador. A máquina utilizada foi então substituída por um PC com processador Xeon 2 GHz, 1 GB de RAM e HD de 80 GB com interface SCSI. Sob esta nova configuração o programa se mostrou estável para transmissões com taxas de até 10 MBps, podendo desta forma ser utilizado para a transmissão dos dados do Simulador de Sinais GPS.

Após os estudos e testes realizados verificou-se que quando o volume de dados a ser transferido é pequeno, podendo ser acomodado na memória livre do computador, a eficiência das transmissões não é muito afetada pelos recursos da máquina utilizada, pois a memória *onboard* disponível na placa é capaz de assegurar a transmissão em alta velocidade. Quando, porém o volume de dados é grande tal que o processador tenha que constantemente acessar o HD e a memória, torna-se clara a necessidade de assegurar que os recursos do computador também sejam suficientes para manter a taxa de transferência desejada.

10. REFERÊNCIAS BIBLIOGRÁFICAS

[1] 653x User Manual, National Instruments

[2] NI-DAQ for PC version 6.9.1 User Manual, National Instruments