Guia do Usuário

Driver do ADS2500

para Delphi, LabVIEW, Matlab e Python

1. INTRODUÇÃO

O *LynxADS2500.LynxDriver* é automation server implementado em Microsoft COM (Component Object Model).

O *LynxADS2500.LynxDriver* pode ser utilizado com linguagens de programação compatíveis com o Microsoft COM, tais como:

- Delphi Tokyo 10.2
- Turbo Delphi
- LabVIEW 2013
- Matlab 2010
- Python

1.1. Requisitos

Para a instalação e utilização do driver são necessários:

- Computador com Windows 7 ou Windows 8.
- 2 GB de memória, recomendamos 4 GB.
- Conhecimento prévio da plataforma de desenvolvimento a ser utilizada: Delphi Tokyo 10.2, LabVIEW, Matlab ou Python.
- Equipamento ADS2500 da Lynx.

1.2. Instalação e Arquivos com Exemplos de Uso

Há dois programas de instalação do driver. Um para instalação no Windows 32 bits e outro para o Windows 64 bits.

O instalador para Windows 64 bits instala e registra o driver nas versões de 32 e 64 bits.

Os arquivos com os fontes dos programas de exemplo de uso são instalados no diretório especificado na instalação do driver. Esses arquivos devem ser utilizados apenas como referência. Uma cópia dos arquivos também é instalada no seguinte subdiretório dos *Documentos Públicos*:

```
Lynx Driver\LynxADS2500\Examples
```

2.1. Formato de Amostras de Canais Analógicos

As amostras dos canais de entradas e saídas digitais são armazenados pelo driver em formato real IEEE de 8 bytes (double) e normalizadas em +/- 1.0. As amostras podem ser lidas pela aplicação no formato normalizado, em volts ou em unidade de engenharia.

Por exemplo, uma amostra obtida de um conversor A/D com faixa de entrada de ±10 volts é normalizada pelo driver de modo que +1.0 corresponde a 10 volts e -1.0 corresponde a -10.0 volts.

Conversão para Unidade de Engenharia
Se no exemplo anterior o canal A/D que está sendo utilizado para medir um sinal de força onde ±10 volts correspondem a ±500 kgf, a conversão do valor da amostra normalizada para valor em unidade de engenharia é dada por:

vEng = vSample * 500

Genericamente,
LimSup: limite superior em unidade de engenharia
LimInf: limite inferior em unidade de engenharia

vEng = (LimSup – LimInf) * (vSample + 1.0) / 2.0 + LimInf

ou

vEng = a * vSample + b

Onde,
a = (LimSup – LimInf) / 2.0
b = a + LimInf

2.2. Formato de Amostras de Canais de Contagem de Pulso

As amostras dos canais de contagem de pulso são armazenadas pelo driver em formato real IEEE de 8 bytes (double) e normalizadas em ±1.0. As amostras podem ser lidas pela aplicação no formato normalizado, em número de pulsos ou em unidade de engenharia.

Conversão para Unidade de Engenharia
A conversão da amostra normalizada de um canal de contagem de pulso depende do modo de operação do contador e do número de bits do contador.

Os modos de operação do contador podem ser divididos em duas categoriais:

- **Contagem com sinal**
- **Contagem sem sinal**
  Essa categoria engloba os modos: medição de período, medição de frequência e contador simples (somente up).

Conversão para Unidade de Engenharia de Contagem com Sinal

$vEng = vSample * 2^{N-1} * Fator$

Onde,

N: número de bits do contador
Fator: valor correspondente a um pulso da contagem

Conversão para Unidade de Engenharia de Contagem sem Sinal

$$vEng = (vSample + 1.0) * 2^{N-1} * Fator$$

Onde,

N: número de bits do contador
Fator: valor correspondente a um pulso da contagem

Observações:

- No modo de medição de período o valor do *Fator* corresponde à resolução na medição de período selecionada no equipamento/driver.
- No modo de medição de frequência o valor do *Fator* corresponde à resolução na medição de frequência selecionada no equipamento/driver.

2.3. Formato de Amostras de Canais Digitais

As amostras dos canais de entradas e saídas digitais são armazenadas em formato inteiro de 32 bits sem sinal, ocupando 4 bytes.

Os bits dos ports de E/S digitais são justificados à direita.

2.4. Amostras em Unidade de Engenharia

As amostras dos canais de entrada analógica e de contagem de pulso também podem ser lidas em unidade de engenharia através dos métodos e propriedades incluídos a partir da versão 2.0 do driver.

3. DESCRIÇÃO DE USO EM DELPHI

O driver do ADS2500 é fornecido com o programa exemplo *TestADS2500* com o código fonte do projeto em Turbo Delphi e Delphi Tokyo 10.2.

3.1. Acesso ao Driver

O módulo ***LynxADS2500_TLB.Pas*** do programa exemplo em Delphi contém a interface com o driver do ADS2500.

Esse módulo deve ser adicionado no projeto em Delphi em que o driver for ser utilizado.

3.1.1. Interface do Driver

O programa deve declarar uma variável para a interface com o driver.

```
LynxDriver: ILynxDriver;
```

A interface *ILynxDriver* está definida no módulo ***LynxADS2500_TLB***.

3.1.2. Instância do Driver

Para acessar o driver do ADS2500, deve-se criar uma instância do Automation Server *LynxADS2500.LynxDriver*.

A instrução abaixo em Pascal cria a instância do driver e atribui a interface do driver na variável *LynxDriver*.

```
LynxDriver := CoLynxDriver.Create;
```

A função CoLynxDriver.Create está definida no módulo ***LynxADS2500_TLB***.

Se o seu programa faz uso de outro driver compatível com o *LynxDriver*, como por exemplo, os drivers do ADS0500, ADS1000 ou ADS1800, a instrução para criação da instância do driver do ADS2500 deverá ser substituída pela instrução:

```
LynxDriver := LynxADS2500_TLB.CoLynxDriver.Create;
```

3.2. Métodos da Interface ILynxDriver

3.2.1. Método Connect

```
Function Connect (Const sPcIP, sHostIP: WideString): WordBool;
```

Este método estabelece a conexão com o ADS2500. Ele deve ser chamado para ter acesso aos demais métodos da interface do driver.

Parâmetro	Descrição
sPcIP	Endereço IP da interface de rede do computador que será utilizada para a comunicação com o equipamento. Se o valor passado neste parâmetro for uma string vazia, a conexão será tentada com uma interface de rede ativa do computador.
sHostIP	Endereço IP do ADS2500. O endereço padrão é '192.168.1.125'.

O método retorna *true* se a conexão foi bem sucedida.

3.2.2. Método Disconnect

```
Function Disconnect: WordBool;
```

Este método finaliza a conexão com o ADS2500.

3.2.3. Método QueryDeviceID

```
Function QueryDeviceID: WordBool;
```

Este método envia a mensagem de Query ID para equipamento. O método retorna *true* se obteve resposta do equipamento.

As informações fornecidas na resposta do equipamento podem ser consultadas através das propriedades *DeviceModel*, *DeviceSN* e *DeviceTag*.

3.2.4. Método ProgCtr

```
Function ProgCtr (Const Channel: smallint;
                  Const Mode, PreScaler: word): WordBool;
```

Este método programa o modo de operação do canal de entrada de contagem de pulso.

Parâmetro	Descrição
Channel	Número do canal de entrada de contagem de pulso. No ADS2500 a numeração de canais é a partir de 1.
Mode	Modo de operação da entrada de contagem de pulso: • 0: medição de frequência • 1: medição de período • 2: contagem de pulso • 4: quadratura (encoder) (disponível apenas para o canal 1)
PreScaler	O ADS2500 não tem suporte para divisor de clock do sinal de entrada. O valor passado neste parâmetro é ignorado pelo driver.

O método retorna *true* se a programação foi realizada com sucesso.

3.2.5. Método ClearICM

```
Procedure ClearICM;
```

Este método limpa a memória de canais de entrada a serem aquisitados. Durante a aquisição os sinais são aquisitados na seguinte ordem:

- Canais de entrada analógica na ordem de inserção na memória de canais.
- Canais de contagem de pulso na ordem de inserção na memória de canais.
- Canais de entrada digital na ordem de inserção na memória de canais.

3.2.6. Método InserICM

```
Function InsertICM (Const ChanType: word;
                    Const Channel: smallint): WordBool;
```

Este método informa o canal de entrada a ser inserido na memória de canais para aquisição de sinais.

Chamadas sucessivas desse método determinam os canais de entrada a serem aquisitados.

Parâmetro	Descrição
ChanType	Tipo do canal de entrada: • 0: canal de entrada analógica • 1: canal de contagem de pulso • 2: canal de entrada digital
Channel	Número do canal de entrada a ser aquisitado. No ADS2500 a numeração de canais é a partir de 1.

O método retorna *true* se a programação foi realizada com sucesso.

3.2.7. Método AcqSetup

```
Function AcqSetup (Const Mode: word;
                   Var FsUser: double): WordBool;
```

Este método é utilizado para a programação do modo da aquisição de sinais.

Parâmetro	Descrição
Mode	Modo da aquisição de sinais: • 0: **Para a aquisição de sinais** Para a aquisição de sinais. • 1: **Inicia a aquisição no modo Polling** Nesse modo de aquisição a frequência de amostragem é definida pelo driver, usualmente 500 Hz, e apenas a última amostra (ou a média das últimas) de cada canal é disponibilizada para a aplicação. Ou seja, as amostras não são armazenadas num buffer circular para leitura periódica da aplicação através do método *GetSamples*. Todos os canais de entrada são aquisitados. • 2: **Inicia a aquisição no modo Single Burst** Nesse modo de aquisição os canais de entrada selecionados são amostrados na taxa de amostragem especificada e o equipamento é programado para enviar um *burst* dos sinais de cada vez. Este modo é apropriado nas aplicações em *LabVIEW* ou *Matlab* em que os sinais de entrada são processados para gerar os sinais de saída a cada intervalo de amostragem. Devido ao tempo de processamento do *LabVIEW* e do *Matlab* em *Windows*, esse modo se aplica às aplicações com passo de loop maiores que 5 ms (200 Hz). • 3: **Inicia a aquisição no modo Multiple Burts** Esse modo de aquisição é análogo ao modo *Single Burst* com a diferença de que o equipamento é programado para enviar as amostras de vários *burts* de cada vez. Esse modo permite uma taxa de amostragem muito mais alta e se aplica aos casos em que não é necessária a atualização dos canais de saída ou a atualização das saídas não precisa ser na mesma taxa dos canais de entrada.
FsUser	Frequência de amostragem dos sinais de entrada. Se a frequência de amostragem especificada não for suportada pelo equipamento, a frequência mais próxima suportada pelo equipamento é utilizada. A frequência de amostragem efetivamente utilizada é retornada nesse parâmetro.

Se o modo de aquisição de sinais for *Single Burst* ou *Multiple Bursts*, o driver programa o equipamento para iniciar a aquisição de sinais. As amostras recebidas do equipamento pelo driver não são armazenadas no buffer circular do driver até que o método *StartCapture* seja executado. Apenas a última amostra de cada canal de entrada habilitado é armazenada pelo driver para ser lida pelos métodos *GetLast*, GetLastEx, *ReadAi*, ReadAiEx, *ReadCtr*, ReadCtrEx, *ReadDi, ReadBitDi* e *ReadBitDiEx*.

O método *AcqSetup* retorna *true* se a operação foi realizada com sucesso.

3.2.8. Método GetSnProp

```
Procedure Connect (Const Signal: smallint;
                   Var sName, sUnit: WideString);
```

Este método informa o nome e a unidade de engenharia de um sinal habilitado (A/D ou entrada de pulso) para aquisição.

Parâmetro	Descrição
Signal	Número de ordem do sinal habilitado para aquisição. A numeração é a partir de 0. Os primeiros sinais correspondentes aos canais de entrada A/D habilitados seguidos dos canais de entrada de pulso habilitados.
sName	Nome do sinal.
sUnit	Unidade de engenharia do sinal.

3.2.9. Método StartCapture

```
Function StartCapture (nSampProg: integer): WordBool;
```

O início da aquisição de sinais propriamente dita é comanda pelo método *AcqSetup*. Após a chamada desse método, o driver passa a receber as amostras do equipamento. No entanto, as amostras recebidas não são armazenadas no buffer circular do driver. Somente a última amostra de cada canal habilitado é salva para ser lido pelo método *GetLast* ou *GetLastEx* e os métodos de leitura de canal de entrada. O método *StartCapture* sinaliza o driver para passar a armazenar as amostras recebidas até atingir o número de amostras por canal programado no parâmetro *nSampProg*. O software aplicativo deve chamar o método *GetSamples* ou *GetSamplesEx* periodicamente para que não ocorra sobreposição no buffer circular do driver.

Parâmetro	Descrição
nSampProg	Número de amostras por canal a serem aquisitadas.

3.2.10. Método StopCapture

```
Function StopCapture: WordBool;
```

Este método finaliza a aquisição de sinais para o buffer circular.

3.2.11. Método GetSignalMap

```
Function GetSignalMap (Var pMap, pDiMap: PSafeArray): WordBool;
```

Este método informa a lista de canais A/D, canais de contagem e ports de entrada digitais habilitados para aquisição.

Parâmetro	Descrição
pMap	Ponteiro para um saferray onde será passada a lista dos canais A/D e canais de contagem habilitados para aquisição. O saferray deve ter as seguintes características: • Tipo de elemento: smallint (int16). • Número de dimensões: 1 ou 2. • Vetor de N elementos ou matriz N x 1 ou 1 x N. • O valor N deve ser maior ou igual ao número de canais A/D e de contagem habilitados para aquisição (propriedades *nAiSignals* e *nCtrSignals*). O número máximo de sinais suportados é igual a 256. • No retorno da função os primeiros *nAiSignals* elementos do safearray correspondem aos canais A/D habilitados. O *nCtrSignals* elementos seguintes correspondem aos canais de contagem habilitados para aquisição. • O valor do elemento do safearray corresponde ao número do canal A/D (ou contagem) correspondente ao índice do safearray.
pDiMap	Ponteiro para um saferray onde será passada a lista dos ports de entrada digital habilitados para aquisição. O saferray deve ter as seguintes características: • Tipo de elemento: smallint (int16). • Número de dimensões: 1 ou 2. • Vetor de N elementos ou matriz N x 1 ou 1 x N. • O valor N deve ser maior ou igual ao número de ports de entrada digital habilitados para aquisição (propriedade *nDiSignals*). O número máximo de ports suportados é igual a 64. • O valor do elemento do safearray corresponde ao número do *port* de entrada digital correspondente ao índice do safearray.

3.2.12. Método GetSamples

```
Function GetSamples (out Status: word;
                     out nSamples, iLast, nSampGot: integer;
                     Var pBuffer, pBufferDI: PSafeArray): WordBool;
```

Este método retorna um conjunto de amostras aquisitadas desde a última chamada do método ou que ainda não foram lidas. O número de amostras retornadas por canal é limitado pelo tamanho dos safearrays.

Este método é utilizado em conjunto com o método *StartCapture*. Na chamada do *StartCapture* é passado o número de amostras a ser aquisitado por canal. As amostras aquisitadas são armazenadas num buffer circular e devem ser lidas periodicamente através do *GetSamples*.

As amostras das entradas analógicas e de contagem de pulso são retornadas em valor normalizado (+/-1.0).

Parâmetro	Descrição
Status	Status da execução do método *GetSamples*. Veja no enumerador *ieStatus* os valores possíveis retornados pelo método.
nSamples	Número de amostras aquisitadas por canal desde o início da aquisição de sinais.
iLast	Índice da última amostra retornada por esta chamada do método.
nSampGot	Número de amostras por canal passadas por esta chamada do método.
pBuffer	Ponteiro para um saferray onde serão passadas as amostras aquisitadas. O saferray deve ter as seguintes características: • Tipo de elemento: Double Os valores das amostras são normalizadas em ±1.0 em relação à faixa de entrada. • Número de dimensões: 1 ou 2. • Vetor de N elementos ou Matriz N x 1 ou 1 x N. o N é o número total de amostras que o vetor ou a matriz comporta. o O número máximo de amostras por canal que o safearray aceita é igual a N div (*nAiSignals* + *nCtrSignals*). • O safearray também pode ser uma Matriz Nt x Ns ou Ns x Nt. o Ns é igual a *nAiSignals* + *nCtrSignals.* o Nt é o número de amostras por canal que o safearray comporta. o A disposição da matriz (Nt x Ns ou Ns x Nt) dependerá da linguagem de programação que irá utilizar o automation server. No Delphi e no C o safearray deve ser definido como uma matriz Nt x Ns, onde Nt é o número de linhas e Ns é o número colunas. No Matlab o safearray deve ser definido como uma matriz Ns x Nt devido ao modo como o Matlab armazena os elementos da matriz.
pBufferDI	Ponteiro para um saferray onde serão passadas as amostras aquisitadas dos ports de entrada digital. O saferray deve ter as seguintes características: • Tipo de elemento: uint32. • Número de dimensões: 1 ou 2. • Vetor de N elementos ou Matriz N x 1 ou 1 x N. o N é o número total de amostras que o vetor ou a matriz comporta. o O número máximo de amostras por canal que o safearray aceita é igual a N div *nDiSignals*. • O safearray também pode ser uma Matriz Nt x Ns ou Ns x Nt. o Ns é igual a *nDiSignals.* o Nt é o número de amostras por canal que o safearray comporta.

Se a função retornar *false* o valor retornado em *Status* deve ser consultado.

3.2.13. Método GetSamplesEx

```
Function GetSamplesEx (out Status: word; SampleFormat: smallint;
                       out nSamples, iLast, nSampGot: integer;
                       Var pBuffer, pBufferDI: PSafeArray): WordBool;
```

Este método retorna um conjunto de amostras aquisitadas desde a última chamada do método ou que ainda não foram lidas. O número de amostras retornadas por canal é limitado pelo tamanho dos safearrays.

Este método é utilizado em conjunto com o método *StartCapture*. Na chamada do *StartCapture* é passado o número de amostras a ser aquisitado por canal. As amostras aquisitadas são armazenadas num buffer circular e devem ser lidas periodicamente através do *GetSamplesEx*.

As amostras das entradas analógicas e de contagem de pulso são retornadas no formato especificado no parâmetro *SampleFormat*.

Parâmetro	Descrição
Status	Status da execução do método *GetSamplesEx*. Veja no enumerador *ieStatus* os valores possíveis retornados pelo método.
SampleFormat	Especifica o formato com o qual as amostras de entrada analógicas e de contagem de pulso serão retornadas: • 0: normalizado (amostras normalizadas em +/- 1.0) • 1: unidade de entrada (volts para entrada analógica e pulsos para entrada de contagem de pulsos) • 2: unidade de engenharia
nSamples	Número de amostras aquisitadas por canal desde o início da aquisição de sinais.
iLast	Índice da última amostra retornada por esta chamada do método.
nSampGot	Número de amostras por canal passadas por esta chamada do método.

pBuffer	Ponteiro para um saferray onde serão passadas as amostras aquisitadas. O saferray deve ter as seguintes características:
	• Tipo de elemento: Double
	• Número de dimensões: 1 ou 2.
	• Vetor de N elementos ou Matriz N x 1 ou 1 x N.
	  ○ N é o número total de amostras que o vetor ou a matriz comporta.
	  ○ O número máximo de amostras por canal que o safearray aceita é igual a N div (*nAiSignals* + *nCtrSignals*).
	• O safearray também pode ser uma Matriz Nt x Ns ou Ns x Nt.
	  ○ Ns é igual a *nAiSignals* + *nCtrSignals.*
	  ○ Nt é o número de amostras por canal que o safearray comporta.
	  ○ A disposição da matriz (Nt x Ns ou Ns x Nt) dependerá da linguagem de programação que irá utilizar o automation server.
	  No Delphi e no C o safearray deve ser definido como uma matriz Nt x Ns, onde Nt é o número de linhas e Ns é o número colunas.
	  No Matlab o safearray deve ser definido como uma matriz Ns x Nt devido ao modo como o Matlab armazena os elementos da matriz.

Parâmetro	Descrição
pBufferDI	Ponteiro para um saferray onde serão passadas as amostras aquisitadas dos ports de entrada digital. O saferray deve ter as seguintes características: • Tipo de elemento: uint32. • Número de dimensões: 1 ou 2. • Vetor de N elementos ou Matriz N x 1 ou 1 x N. o N é o número total de amostras que o vetor ou a matriz comporta. o O número máximo de amostras por canal que o safearray aceita é igual a N div *nDiSignals*. • O safearray também pode ser uma Matriz Nt x Ns ou Ns x Nt. o Ns é igual a *nDiSignals.* o Nt é o número de amostras por canal que o safearray comporta.

Se a função retornar *false* o valor retornado em *Status* deve ser consultado.

3.2.14. Método GetLast

```
Function GetLast (out Status: word;
                  Var pBuffer, pBufferDI: PSafeArray): WordBool;
```

Este método retorna o valor da última amostra aquisitada de cada sinal de entrada. As amostras das entradas analógicas e de contagem de pulso são retornadas em valor normalizado (+/-1.0).

Parâmetro	Descrição
Status	Status da execução do método *GetLast*. Veja no enumerador *ieStatus* os valores possíveis retornados pelo método.
pBuffer	Ponteiro para um saferray onde serão passadas as amostras aquisitadas. O saferray deve ter as seguintes características: • Tipo de elemento: double. Os valores das amostras são normalizadas em ±1.0 em relação à faixa de entrada • Número de dimensões: 1 ou 2. • Vetor de N elementos ou Matriz N x 1 ou 1 x N. N deve ser maior ou igual a *nAiSignals* + *nCtrSignals*.
pBufferDI	Ponteiro para um saferray onde serão passadas as amostras aquisitadas dos ports de entrada digital. O saferray deve ter as seguintes características: • Tipo de elemento: uint32. • Número de dimensões: 1 ou 2. • Vetor de N elementos ou Matriz N x 1 ou 1 x N. N deve ser maior ou igual *nDiSignals*.

Se a função retornar *false* o valor retornado em *Status* deve ser consultado.

3.2.15. Método GetLastEx

```
Function GetLastEx (out Status: word; SampleFormat: smallint;
                    Var pBuffer, pBufferDI: PSafeArray): WordBool;
```

Este método retorna o valor da última amostra aquisitada de cada sinal de entrada. As amostras das entradas analógicas e de contagem de pulso são retornadas no formato especificado no parâmetro *SampleFormat*.

Parâmetro	Descrição
Status	Status da execução do método *GetLastEx*. Veja no enumerador *ieStatus* os valores possíveis retornados pelo método.
SampleFormat	Especifica o formato com o qual as amostras de entrada analógicas e de contagem de pulso serão retornadas: • 0: normalizado (amostras normalizadas em +/- 1.0) • 1: unidade de entrada (volts para entrada analógica e pulsos para entrada de contagem de pulsos) • 2: unidade de engenharia
pBuffer	Ponteiro para um saferray onde serão passadas as amostras aquisitadas. O saferray deve ter as seguintes características: • Tipo de elemento: double. • Número de dimensões: 1 ou 2. • Vetor de N elementos ou Matriz N x 1 ou 1 x N. N deve ser maior ou igual a *nAiSignals* + *nCtrSignals*.
pBufferDI	Ponteiro para um saferray onde serão passadas as amostras aquisitadas dos ports de entrada digital. O saferray deve ter as seguintes características: • Tipo de elemento: uint32. • Número de dimensões: 1 ou 2. • Vetor de N elementos ou Matriz N x 1 ou 1 x N. N deve ser maior ou igual *nDiSignals*.

Se a função retornar *false* o valor retornado em *Status* deve ser consultado.

3.2.16. Método ReadAi

```
Function ReadAi (Const Channel: smallint): double;
```

Este método retorna o valor da última amostra lida do canal A/D especificado. O valor da amostra é normalizado em ±1.0 (consulte o tópico 2 deste guia do usuário).

Parâmetro	Descrição
Channel	Número do canal A/D do equipamento. No ADS2500 a numeração dos canais é a partir de 1. Se o canal A/D não estiver habilitado ou a aquisição de sinais não estiver ativa, o método valor zero.

3.2.17. Método ReadAiEx

```
Function ReadAiEx (Const Channel, SampleFormat: smallint): double;
```

Este método retorna o valor da última amostra lida do canal A/D especificado.

Parâmetro	Descrição
Channel	Número do canal A/D do equipamento. No ADS2500 a numeração dos canais é a partir de 1. Se o canal A/D não estiver habilitado ou a aquisição de sinais não estiver ativa, o método valor zero.
SampleFormat	Especifica o formato com o qual a amostra da entrada analógica será retornada: • 0: normalizado (amostras normalizadas em +/- 1.0) • 1: unidade de entrada (volts) • 2: unidade de engenharia

3.2.18. Método ReadCtr

```
Function ReadCtr (Const Channel: smallint): double;
```

Este método retorna o valor da última amostra lida do canal de contagem de pulso. O valor da amostra é normalizado em ±1.0 (consulte o tópico 2 deste guia do usuário).

Parâmetro	Descrição
Channel	Número do canal de contagem de pulso. No ADS2500 a numeração dos canais é a partir de 1. Se o canal de contagem de pulso não estiver habilitado ou a aquisição de sinais não estiver ativa, o método valor zero.

3.2.19. Método ReadCtrEx

```
Function ReadCtrEx (Const Channel, SampleFormat: smallint): double;
```

Este método retorna o valor da última amostra lida do canal de contagem de pulso.

Parâmetro	Descrição
Channel	Número do canal de contagem de pulso. No ADS2500 a numeração dos canais é a partir de 1. Se o canal de contagem de pulso não estiver habilitado ou a aquisição de sinais não estiver ativa, o método valor zero.
SampleFormat	Especifica o formato com o qual a amostra da entrada de contagem de pulso será retornada: • 0: normalizado (amostras normalizadas em +/- 1.0) • 1: unidade de entrada (pulsos) • 2: unidade de engenharia

3.2.20. Método ReadDi

```
Function ReadDi (Const Channel: smallint): longword;
```

Este método retorna o valor da última amostra lida do canal de entrada digital.

Parâmetro	Descrição
Channel	Número do canal de entrada digital. No ADS2500 a numeração dos canais é a partir de 1. Se o canal de entrada digital não estiver habilitado ou a aquisição de sinais não estiver ativa, o método valor zero.

3.2.21. Método ReadBitDi

```
Function ReadBitDi (Const Channel, Bit: smallint): WordBool;
```

Este método é utilizado para a leitura de um bit de um canal de entrada digital. O método retorna *true* se o bit estiver em nível lógico 1 ou *false* se estiver em nível lógico 0.

Parâmetro	Descrição
Channel	Número do canal de entrada digital. No ADS2500 a numeração dos canais é a partir de 1.
Bit	Bit do canal de entrada digital a ser lido.

3.2.22. Método ReadBitDiEx

```
Function ReadBitDiEx (Const Channel, Bit: smallint): WideString;
```

Este método é utilizado para a leitura de um bit de um canal de entrada digital. O método retorna o texto correspondente ao nível lógico 0 e 1 definidos respectivamente pelas propriedades *DiBitLow* e *DiBitHigh*.

Parâmetro	Descrição
Channel	Número do canal de entrada digital. No ADS2500 a numeração dos canais é a partir de 1.
Bit	Bit do canal de entrada digital a ser lido.

3.2.23. Método ReadDo

```
Function ReadDo (Const Channel: smallint): longword;
```

Este método retorna o valor da última amostra escrita no canal de saída digital.

Parâmetro	Descrição
Channel	Número do canal de saída digital. No ADS2500 a numeração dos canais é a partir de 1.

3.2.24. Método ReadBitDo

```
Function ReadBitDo (Const Channel, Bit: smallint): WordBool;
```

Este método é utilizado para a leitura de um bit de um canal de saída. O método retorna *true* se o bit estiver em nível lógico 1 ou *false* se estiver em nível lógico 0.

Parâmetro	Descrição
Channel	Número do canal de saída digital. No ADS2500 a numeração dos canais é a partir de 1.
Bit	Bit do canal de saída digital a ser lido.

3.2.25. Método ReadBitDoEx

```
Function ReadBitDoEx (Const Channel, Bit: smallint): WideString;
```

Este método é utilizado para a leitura de um bit de um canal de saída. O método retorna o texto correspondente ao nível lógico 0 e 1 definidos respectivamente pelas propriedades *DoBitLow* e *DoBitHigh*.

Parâmetro	Descrição
Channel	Número do canal de saída digital. No ADS2500 a numeração dos canais é a partir de 1.
Bit	Bit do canal de saída digital a ser lido.

3.2.26. Método WriteDo

```
Procedure WriteDo (Const Channel: smallint; Const Value: longword);
```

Este método é utilizado para a escrita em canal de saída digital.

Parâmetro	Descrição
Channel	Número do canal de saída digital. No ADS2500 a numeração dos canais é a partir de 1.
Value	Valor a ser escrito no canal de saída digital.

3.2.27. Método WriteBitDo

```
Procedure WriteBitDo (Const Channel, Bit: smallint; Value: WordBool);
```

Este método é utilizado para a escrita em um bit de um canal de saída digital.

Parâmetro	Descrição
Channel	Número do canal de saída digital. No ADS2500 a numeração dos canais é a partir de 1.
Bit	Bit do canal de saída digital a ser lido.
Value	O valor a ser escrito no bit, *true* para setar e *false* para resetar.

3.2.28. Método WriteAo

```
Procedure WriteAo (Const Channel: smallint; Const Value: double);
```

Este método é utilizado para a escrita em canal de saída analógica.

Parâmetro	Descrição
Channel	Número do canal de saída analógica. No ADS2500 a numeração dos canais é a partir de 1.
Value	Valor a ser escrito no canal de saída analógica. O valor da amostra é normalizado em ±1.0. Por exemplo, se a saída analógica estiver configurada para operar em ±10 volts, O valor normalizado 1.0 corresponderá a +10 volts e o valor normalizado -1.0 corresponderá a -10 volts.

3.2.29. Método WriteAoEx

```
Procedure WriteAoEx (Const Channel, SampleFormat: smallint;
                     Const Value: double);
```

Este método é utilizado para a escrita em canal de saída analógica.

Parâmetro	Descrição
Channel	Número do canal de saída analógica. No ADS2500 a numeração dos canais é a partir de 1.
SampleFormat	Especifica o formato do valor da amostra da saída analógica especificado no parâmetro *Value*: • 0: normalizado (amostras normalizadas em +/- 1.0) • 1: unidade de saída (volts ou mA) • 2: unidade de engenharia
Value	Valor a ser escrito no canal de saída analógica.

3.2.30. Método SetAiSetup

```
Procedure SetAiSetup (Const Channel: smallint; sName, sUnit: WideString;
                      HiLim, LoLim: double);
```

Este método equivale à atribuição das propriedades *AiName*, *AiUnit*, *AiHiLim* e *AiLoLim* de um canal de entrada analógica.

Parâmetro	Descrição
Channel	Número do canal de entrada analógica. Numeração a partir de 1.
sName	Nome do sinal.
sUnit	Unidade de engenharia.
HiLim	Limite superior em unidade de engenharia.
LoLim	Limite inferior em unidade de engenharia.

3.2.31. Método GetAiSetup

```
Procedure GetAiSetup (Const Channel: smallint;
                      Var sName, sUnit: WideString;
                      Var HiLim, LoLim: double);
```

Este método equivale à leitura das propriedades *AiName*, *AiUnit*, *AiHiLim* e *AiLoLim* de um canal de entrada analógica.

Parâmetro	Descrição
Channel	Número do canal de entrada analógica. Numeração a partir de 1.
sName	Nome do sinal.
sUnit	Unidade de engenharia.
HiLim	Limite superior em unidade de engenharia.
LoLim	Limite inferior em unidade de engenharia.

3.2.32. Método SetAoSetup

```
Procedure SetAoSetup (Const Channel: smallint; sName, sUnit: WideString;
                      HiLim, LoLim: double);
```

Este método equivale à atribuição das propriedades *AoName*, *AoUnit*, *AoHiLim* e *AoLoLim* de um canal de saída analógica.

Parâmetro	Descrição
Channel	Número do canal de saída analógica. Numeração a partir de 1.
sName	Nome do sinal.
sUnit	Unidade de engenharia.
HiLim	Limite superior em unidade de engenharia.
LoLim	Limite inferior em unidade de engenharia.

3.2.33. Método GetAoSetup

```
Procedure GetAoSetup (Const Channel: smallint;
                      Var sName, sUnit: WideString;
                      Var HiLim, LoLim: double);
```

Este método equivale à leitura das propriedades *AoName*, *AoUnit*, *AoHiLim* e *AoLoLim* de um canal de saída analógica.

Parâmetro	Descrição
Channel	Número do canal de saída analógica. Numeração a partir de 1.
sName	Nome do sinal.
sUnit	Unidade de engenharia.
HiLim	Limite superior em unidade de engenharia.
LoLim	Limite inferior em unidade de engenharia.

3.2.34. Método SetCtrSetup

```
Procedure SetCtrSetup (Const Channel: smallint;
                       sName, sUnit: WideString; Factor: double);
```

Este método equivale à atribuição das propriedades *CtrName*, *CtrUnit* e *CtrFactor* de um canal de entrada de pulso.

Parâmetro	Descrição
Channel	Número do canal de entrada de pulso. Numeração a partir de 1.
sName	Nome do sinal.
sUnit	Unidade de engenharia.
CtrFactor	Fator de conversão do sinal no canal de contagem de pulso para unidade de engenharia. Valor em unidade de engenharia por pulso.

3.2.35. Método GetCtrSetup

```
Procedure GetCtrSetup (Const Channel: smallint;
                       Var sName, sUnit: WideString;
                       Var Factor: double);
```

Este método equivale à leitura das propriedades *CtrName*, *CtrUnit* e *CtrFactor* de um canal de entrada de pulso.

Parâmetro	Descrição
Channel	Número do canal de entrada de pulso. Numeração a partir de 1.
sName	Nome do sinal.
sUnit	Unidade de engenharia.
CtrFactor	Fator de conversão do sinal no canal de contagem de pulso para unidade de engenharia. Valor em unidade de engenharia por pulso.

3.3. Enumerador ieStatus

O enumerador *ieStatus* é utilizado no parâmetro *Status* dos métodos *GetSamples* e *GetLast* e na propriedade *LastErrorCode*.

ieStatus	Valor	Descrição
ieNoError	0	Nenhum erro.
ieBufOverrun	1	Ocorreu overrun no buffer de aquisição de sinais da thread.
ieAcqOff	2	Aquisição de sinais não está ativa.
ieWinsock	3	Indica erro do winsock do Windows
ieHostClosed	4	Indica que o equipamento finalizou a conexão
ieNotConnected	5	Indica que não está conectado com o equipamento
ieInvSafearray	6	Indica safearray inválido. Pode ser dimensões erradas ou tipo incompatível de elemento do array.
ieStopNotif	7	Ocorreu uma notificação de parada da aquisição de sinais informada pelo equipamento. A aplicação cliente deve reiniciar a conexão posteriormente.
ieAbortNotif	8	Ocorreu uma notificação de conexão abortada pelo equipamento por inatividade da aplicação. A aplicação cliente deve reiniciar a conexão.
ieAcqDone	9	Indica que a aquisição do número de amostras programado pela chamada do método *StartCapture* foi completada.
ecInvChannel	101	Canal inválido
ecInvChanType	102	Tipo de canal inválido.
ecICMfull	103	Memória de canais está cheio
ecInvCtrMode	104	Modo de operação inválido para o canal de contagem de pulso.
ecInvAcqMode	105	Modo de aquisição inválido.
ecInvSFormat	151	Formato de amostra inválido
ecNoSignal	106	Nenhum canal habilitado para aquisição
ecInvSFormat	156	Formato de amostra inválido
ecInvAddrIP	201	Endereço IP inválido
ecWSAbind	202	Erro no bind do winsock
ecWSAconnect	203	Erro no connect do winsock
ecWSArecv	204	Erro no recv do winsock
ecWSAsend	205	Erro no send do winsock
ecNoResponse	206	Sem resposta do equipamento
ecThreadTmout	207	Timeout da thread da comunicação com o equipamento

3.4. Propriedades da Interface ILynxDriver

Propriedade	Type	R/W	Index	Descrição
LastErrorCode	short smallint	R		Código do último erro
IsConnected	VARIANT_BOOL WordBool	R		Indica se está conectado com os equipamentos
sPcIP	BSTR WideString	R		Endereço IP da interface de rede do computador
sHostIP	BSTR WideString	R		Endereço IP do equipamento
DeviceModel	BSTR WideString	R		Modelo do hardware de aquisição de dados
DeviceSN	BSTR WideString	R		Número de série do hardware
DeviceTag	BSTR WideString	R		Tag do hardware
EnumStart1 [1]	VARIANT_BOOL WordBool	R		Indica se a numeração dos canais se inicia em 1.
nAiChannels	short smallint	R		Número de canais A/D
nAiRange	short smallint	R		Número de opções de faixa de entrada do conversor A/D
AiRange	double double	R	Index 0 a nAiRange-1	Valor da Index-ésima faixa de entrada do conversor A/D. Valor positivo indica faixa unipolar. Por exemplo, o valor 10 indica uma faixa de entrada de 0 a 10. Valor negativo indica faixa bipolar. Por exemplo, o valor -10 indica uma faixa de entrada de -10 a 10.
AiRangeUnit	BSTR WideString	R	Index 0 a nAiRange-1	Unidade da Index-ésima faixa do conversor A/D
AiRangeIndex	unsigned short word	R/W		Seleciona a faixa de entrada do conversor A/D
nAoChannels	short smallint	R		Número de canais D/A
nAoRange	short smallint	R		Número de opções de faixa de faixa de saída do conversor D/A
AoRange	double double	R	Index 0 a nAoRange-1	Valor da Index-ésima faixa de saída do conversor D/A. Valor positivo indica faixa unipolar. Por exemplo, o valor 10 indica uma faixa de saída de 0 a 10. Valor negativo indica faixa bipolar. Por exemplo, o valor -10 indica uma faixa de saída de -10 a 10.
AoRangeUnit	BSTR WideString	R	Index 0 a nAoRange-1	Unidade da Index-ésima faixa de saída do conversor D/A
AoRangeIndex	unsigned short word	R/W		Seleciona a faixa de saída do conversor D/A
nCtrChannels	short smallint	R		Número de canais de contagem
nCtrBits	short Smallint	R		Número de bits dos canais de contagem
nFResol	short smallint	R		Número de opções de resolução na medição de frequência nos canais de contagem de pulso.
FResol	double double	R	Index 0 a FResol-1	Valor da Index-ésima resolução na medição de frequência nos canais de contagem de pulso (em Hz)
FResolIndex	unsigned short word	R/W		Seleciona a resolução da medição de frequência nos canais de contagem de pulso

Propriedade	Type	R/W	Index	Descrição
nTResol	short smallint	R		Número de opções de resolução na medição de período nos canais de contagem de pulso.
TResol	double double	R	Index 0 a TResol-1	Valor da Index-ésima resolução da medição de período nos canais de contagem de pulso (em microsegundos)
TResolIndex	unsigned short word	R/W		Seleciona a resolução da medição de período nos canais de contagem de pulso
nDiPorts	short smallint	R		Número de ports de entrada digital
nDiBits	short smallint	R		Número de bits por port entrada digital
nDoPorts	short smallint	R		Número de ports de saída digital
nDoBits	short smallint	R		Número de bits por port de saída digital
IsAcqOn	VARIANT_BOOL WordBool	R		Indica se a aquisição de sinais está ativa
SampleFreq	double	R		Frequência de amostragem
nAiSignals	short smallint	R		Número de canais A/D habilitados para aquisição
nCtrSignals	short smallint	R		Número de canais de contagem habilitados para aquisição
nDiSignals	short smallint	R		Número de ports DI habilitados para aquisição
tSample	__int64 int64	R		Número de amostras aquisitadas desde o início da aquisição após a chamada do método *AcqSetup* com comando de partida da aquisição. Equivale ao contador de intervalos de amostragem e pode ser útil para determinar quando há um dado novo nas aplicações que utilizam o modo de aquisição *Single Burst*.
nSampProg	integer long	R		Número de amostras a serem aquisitadas, programada no método *StartCapture*
nSamples	integer long	R		Número de amostras aquisitadas por canal do total de *nSampProg*
AiName [1, 2, 3]	BSTR WideString	R/W	Channel	Nome do sinal canal de entrada analógica
AiUnit [1, 2, 3]	BSTR WideString	R/W	Channel	Unidade de engenharia do sinal no canal de entrada analógica
AiHiLim [1, 2, 3]	double double	R/W	Channel	Limite superior do sinal no canal de entrada analógica em unidade de engenharia
AiLoLim [1, 2, 3]	double double	R/W	Channel	Limite inferior do sinal no canal de entrada analógica em unidade de engenharia
CtrName [1, 2, 3]	BSTR WideString	R/W	Channel	Nome do sinal no canal de entrada de pulso
CtrUnit [1, 2, 3]	BSTR WideString	R/W	Channel	Unidade de engenharia do sinal no canal de entrada de pulso
CtrFactor [1, 2, 3]	double double	R/W	Channel	Fator de conversão do sinal no canal de contagem de pulso para unidade de engenharia. Valor em unidade de engenharia por pulso.
DiBitName [1, 2, 3]	BSTR WideString	R/W	Channel, Bit	Nome do bit de entrada digital
DiBitLow [1, 2, 3]	BSTR WideString	R/W	Channel, Bit	Valor para nível low de bit de entrada digital
DiBitHigh [1, 2, 3]	BSTR WideString	R/W	Channel, Bit	Valor para nível high de bit de entrada digital
AoName [1, 2, 3]	BSTR WideString	R/W	Channel	Nome do sinal no canal de saída analógica
AoUnit [1, 2, 3]	BSTR WideString	R/W	Channel	Unidade de engenharia do sinal no canal de saída analógica

Propriedade	Type	R/W	Index	Descrição
AoHiLim [1, 2, 3]	double double	R/W	Channel	Limite superior do canal de saída analógica em unidade de engenharia
AoLoLim [1, 2, 3]	double double	R/W	Channel	Limite inferior do canal de saída analógica em unidade de engenharia
DoBitName [1, 2, 3]	BSTR WideString	R/W	Channel, Bit	Nome do bit de saída digital
DoBitLow [1, 2, 3]	BSTR WideString	R/W	Channel, Bit	Valor para nível low de bit de saída digital
DoBitHigh [1, 2, 3]	BSTR WideString	R/W	Channel, Bit	Valor para nível high de bit de saída digital

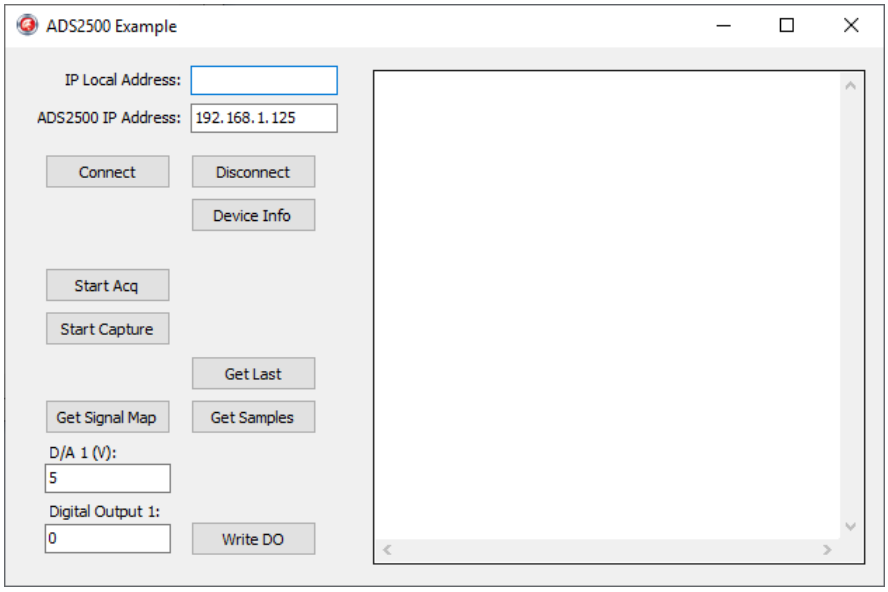
Notas:

1. Essas propriedades foram incluídas na versão 2.0 da interface *ILynxDriver*.
2. Os valores dessas propriedades são carregados do ADS2500 durante a chamada do método *Connect*. É mais fácil utilizar o assistente do *ADS2500* através do *Lynx@Net* para configurar os canais. Após a configuração ter sido salva no ADS2500, ela é carregada pelo driver através do método *Connect*.
3. A alteração dos valores dessas propriedades só tem efeito durante a operação on line.

3.5. Programa Exemplo em Delphi Tokyo 10.2

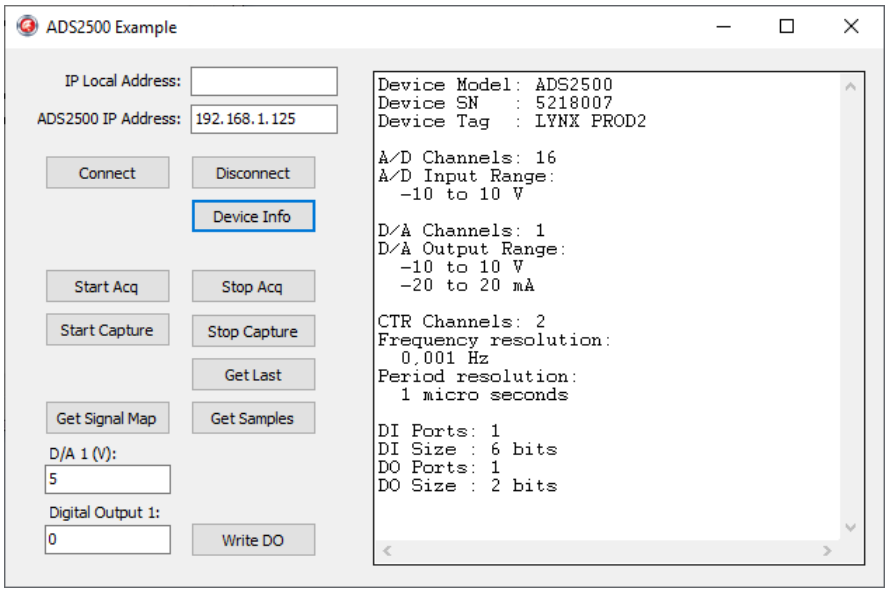
O driver do ADS2500 é fornecido com o programa exemplo *TestADS2500* com código fonte em Delphi Tokyo 10.2 e em Turbo Delphi.

A figura seguinte apresenta a janela do programa de teste.



A janela do programa *TestADS2500* apresenta os seguintes botões:

- **Connect**
  Estabelece a conexão com o ADS2500 com os endereços IP especificados nos campos *IP Local Address* e *ADS2500 IP Address*.
- **Disconnect**
  Finaliza a conexão com o ADS2500.
- **Device Info**
  Executa o método *QueryDeviceID* e apresenta algumas informações do ADS2500 consultadas através das propriedades acessadas pela interface do driver. As informações são listadas no Memo da janela do programa.



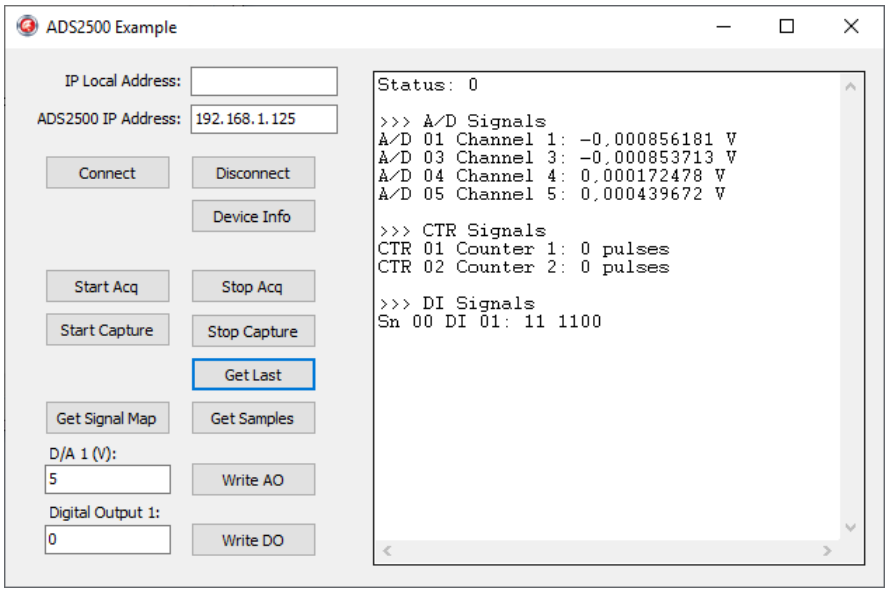
- **Start Acq**

Programa e inicia a aquisição de sinais.
É programada a aquisição dos canais A/D 1, 3, 4 e 5, os dois canais de contagem de pulso e o port de entrada digital. A frequência de amostragem é programada para 8 kHz.

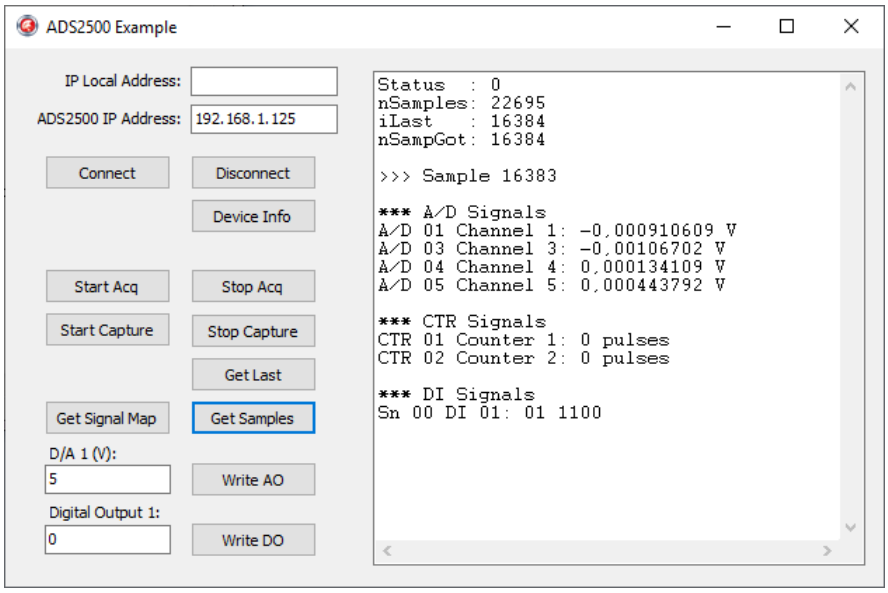
- **Stop Acq**
  Finaliza a aquisição de sinais.

- **Start Capture**
  Programa a captura dos canais de entrada habilitados para aquisição. As amostras dessas entradas são armazenadas no buffer circular do driver e são disponibilizados para leitura pela aplicação através de chamadas periódicas do método *GetSamplesEx*. A captura é finalizada automaticamente pelo driver quando o número de amostras especificados na chamada do método *StartCapture* for atingido.

- **Get Last**
  Executa o método *GetLastEx* para obter a última amostra aquisitada de cada canal de entrada habilitado para aquisição. Na chamada do método é especificada a leitura em unidade de engenharia. Os valores das amostras são apresentados no Memo da janela.



❖ **Get Samples**
Executa o método *GetSamplesEx* para obter um conjunto de amostras armazenadas pelo driver no buffer circular. Na chamada do método é especificada a leitura em unidade de engenharia. A última amostra de cada canal retornada nessa chamada é apresentada no Memo da janela.



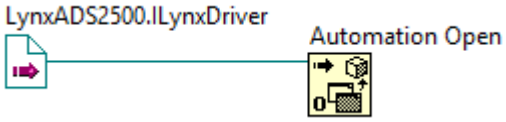
❖ **Get Signal Map**
Executa o método *GetSignalMap* para obter a lista dos canais de entrada habilitados para aquisição.

❖ **Write AO**
Escreve o valor especificado no campo ***D/A 1 (V)*** no canal de saída analógica do ADS2500.

❖ **Write DO**
Escreve o valor especificado no campo ***Digital Output 1*** no port de saída digital do ADS2500.

4.1. Automation Open

A instância do driver do ADS2500 é criada através do bloco ***Automation Open*** do LabVIEW.



Para inserir o bloco *Automation Open*, siga os seguintes passos:

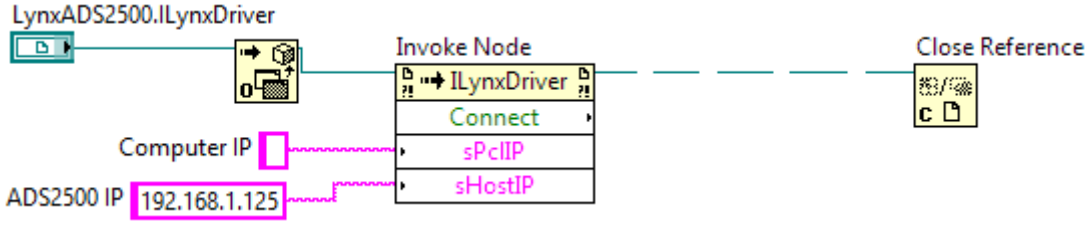
1. Se a janela *Functions* do LabVIEW não estiver visível, selecione *View* / *Functions*.
2. Na janela *Functions*, abra a paleta *Connectivity* e clique em *ActiveX*.
3. Na paleta *Connectivity* / *ActiveX*, selecione ***Automation Open*** e arraste para o seu diagrama de bloco.
4. Crie uma constante para o terminal *automation refnun* do bloco.
5. Clique com o botão direito do mouse sobre o bloco da constante criada no passo anterior e selecione o menu *Select AtiveX Class* no popup menu apresentado. Selecione a classe *LynxADS2500.ILynxDriver*.
6. Se a classe *LynxADS2500.ILynxDriver* não estiver na lista do passo anterior, selecione *Browse*.
7. Na janela *Select Object From Type Library* aberta, clique no combo box *Type Library*, selecione *LynxADS2500* e pressione o botão *OK*. Retorne para o passo 5.

Após criar a instância do *LynxADS2500.ILynxDriver*, tem-se acesso às propriedades e métodos da interface *ILynxDriver* implementada pelo *automation LynxADS2500.ILynxDriver*.

A saída do bloco *Automation Open* é um *handle* que deve ser conectado na entrada *reference* dos blocos de chamadas de métodos e propriedades da interface *ILynxDriver*.

4.2. Close Reference

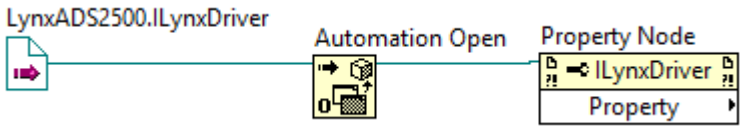
Para cada bloco *Automation Open* incluído no programa em *LabVIEW* deve ser incluída uma chamada do bloco ***Close Reference*** para finalizar a instância do automation quando ele não for mais necessário.



O bloco *Close Reference* se encontra na paleta de funções *Connectivity* / *ActiveX*. A entrada *reference* desse bloco deve ser conectada no terminal *reference out* de um bloco *Invoke Node* ou *Property Node* associado ao automation *LynxADS2500.ILynxDriver*.

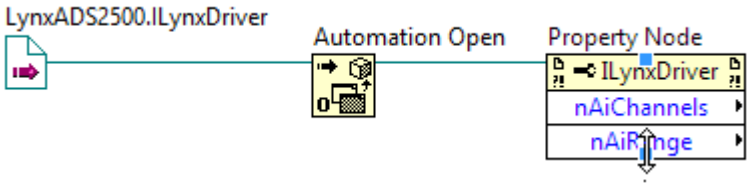
4.3. Acesso às Propriedades da Interface ILynxDriver

As propriedades da interface *ILynxDriver* são acessadas através do bloco ***Property Node*** disponível na paleta de funções *Connectivity / ActiveX* da janela *Functions* do LabVIEW.



Para acessar uma propriedade da interface *ILynxDriver*:

1. Selecione o bloco de função ***Property Node*** na paleta *Connectivity / ActiveX*, da janela *Functions* e arraste para o seu diagrama de bloco.
2. Conecte o terminal de entrada ***reference*** no terminal de saída *Automation Refnum* do bloco *Automation Open* ou no terminal *reference out* de um bloco *Property Node* ou *Invoke Node* que esteja referenciado à interface *ILynxDriver*. Veja a figura anterior.
3. Clique no campo ***Property*** do bloco *Property Node* inserido e selecione a propriedade a ser acessada.
4. Para incluir uma nova propriedade no mesmo bloco, clique com o botão direito do mouse sobre o campo de propriedade do bloco e selecione o comando ***Add Element*** no popup menu apresentado. Outra maneira é dimensionar o tamanho do bloco pela parte inferior, como ilustrado na figura abaixo.



A tabela seguinte lista as propriedades da interface *ILynxDriver* que podem ser acessadas através do bloco *Property Node*.

Propriedade	Type	R/W	Descrição
LastErrorCode	I16	R	Código do último erro
IsConnected	Boolean	R	Indica se está conectado com os equipamentos
sPcIP	String	R	Endereço IP da interface de rede do computador
sHostIP	String	R	Endereço IP do equipamento
DeviceModel	String	R	Modelo do hardware de aquisição de dados
DeviceSN	String	R	Número de série do hardware
DeviceTag	String	R	Tag do hardware
EnumStart1 [1]	Boolean	R	Indica se a numeração dos canais se inicia em 1
nAiChannels	I16	R	Número de canais A/D
nAiRange	I16	R	Número de opções de faixa de entrada do conversor A/D
AiRangeIndex	U16	R/W	Seleciona a faixa de entrada do conversor A/D
nAoChannels	I16	R	Número de canais D/A
nAoRange	I16	R	Número de opções de faixa de faixa de saída do conversor D/A
AoRangeIndex	U16	R/W	Seleciona a faixa de saída do conversor D/A
nCtrChannels	I16	R	Número de canais de contagem
nCtrBits	I16	R	Número de bits dos canais de contagem

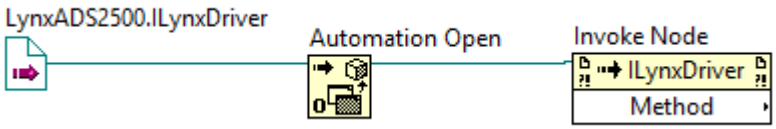
Propriedade	Type	R/W	Descrição
nFResol	I16	R	Número de opções de resolução na medição de frequência nos canais de contagem de pulso.
FResolIndex	U16	R/W	Seleciona a resolução da medição de frequência nos canais de contagem de pulso
nTResol	I16	R	Número de opções de resolução na medição de período nos canais de contagem de pulso.
TResolIndex	U16	R/W	Seleciona a resolução da medição de período nos canais de contagem de pulso
nDiPorts	I16	R	Número de ports de entrada digital
nDiBits	I16	R	Número de bits por port entrada digital
nDoPorts	I16	R	Número de ports de saída digital
nDoBits	I16	R	Número de bits por port de saída digital
IsAcqOn	Boolean	R	Indica se a aquisição de sinais está ativa
SampleFreq	DBL	R	Frequência de amostragem
nAiSignals	I16	R	Número de canais A/D habilitados para aquisição
nCtrSignals	I16	R	Número de canais de contagem habilitados para aquisição
nDiSignals	I16	R	Número de ports DI habilitados para aquisição
tSample	I64	R	Número de amostras aquisitadas desde o início da aquisição após a chamada do método *AcqSetup* com comando de partida da aquisição. Equivale ao contador de intervalos de amostragem e pode ser útil para determinar quando há um dado novo nas aplicações que utilizam o modo de aquisição *Single Burst*.
nSampProg	I32	R	Número de amostras a serem aquisitadas, programada no método *StartCapture*
nSamples	I32	R	Número de amostras aquisitadas por canal do total de *nSampProg*

Nota:

1. Propriedade incluída na versão 2.0 da interface *ILynxDriver*.

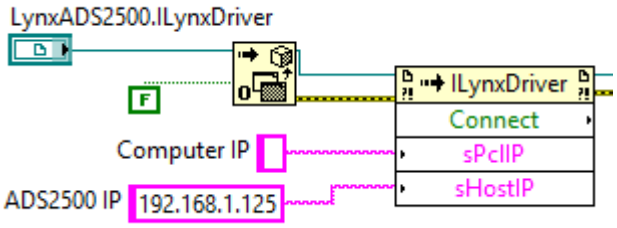
4.4. Acesso aos Métodos da Interface ILynxDriver

Os métodos da interface *ILynxDriver* são acessados através do bloco ***Invoke Node*** disponível na paleta de funções *Connectivity / ActiveX* da janela *Functions* do LabVIEW.



Para acessar um método da interface *ILynxDriver*:

1. Selecione o bloco de função ***Invoke Node*** na paleta *Connectivity / ActiveX*, da janela *Functions* e arraste para o seu diagrama de bloco.
2. Conecte o terminal de entrada ***reference*** no terminal de saída *Automation Refnum* do bloco *Automation Open* ou no terminal *reference out* de um bloco *Property Node* ou *Invoke Node* que esteja referenciado à interface *ILynxDriver*. Veja a figura anterior.
3. Clique no campo ***Method*** do bloco ***Invoke*** *Node* inserido e selecione o método a ser chamado.
4. Após selecionar um método da interface *ILynxDriver*, o bloco é atualizado e passa a apresentar terminais de entrada e saída que correspondem aos parâmetros de entrada e saída do método. Se o método for uma função, um terminal de saída é apresentado no campo com o nome do método. A figura abaixo ilustra um exemplo com a chamada do método *Connect* da interface *ILynxDriver*.



A tabela seguinte lista os métodos da interface *ILynxDriver* que podem ser acessadas através do bloco *Invoke Node*.

Função	Descrição
Connect	Estabelece a conexão o ADS2500.
Disconnect	Fecha a conexão com o ADS2500.
QueryDeviceID	Obtém identificação do equipamento
ProgCtr	Programa modo de operação de canal de contagem de pulso
ClearICM	Limpa a memória de canais
InsertICM	Insere canal na memória de canais
AcqSetup	Configura a aquisição de sinais
StartCapture	Inicia a captura da aquisição de sinais
StopCapture	Finaliza a captura da aquisição de sinais
GetSignalMap	Mapeamento dos sinais aquisitados com os canais físicos
GetSamples	Leitura das últimas amostras aquisitadas de cada sinal
GetSamplesEx [1]	Leitura das últimas amostras aquisitadas de cada sinal
GetLast	Leitura da última amostra aquisitada de cada sinal
GetLastEx [1]	Leitura da última amostra aquisitada de cada sinal
ReadAi	Leitura de canal de entrada analógica
ReadAiEx [1]	Leitura de canal de entrada analógica
ReadCtr	Leitura de canal de contagem de pulso
ReadCtrEx [1]	Leitura de canal de contagem de pulso
ReadDi	Leitura de port de entrada digital
ReadBitDi	Leitura de bit de port de entrada digital
ReadBitDiEx [1]	Leitura de bit de port de entrada digital
ReadDo	Leitura de port de saída digital (última escrita)
ReadBitDo	Leitura de bit de port de saída digital
ReadBitDoEx [1]	Leitura de bit de port de saída digital
WriteDo	Escrita em port de saída digital
WriteBitDo	Escrita de bit de port de saída digital
WriteAo	Escrita em canal de saída analógica
WriteAoEx [1]	Escrita em canal de saída analógica
GetSnProp [1]	Informa o nome e a unidade de engenharia de um sinal habilitado para aquisição de sinais (canal A/D ou de contagem de pulso)
SetAiSetup [1]	Especifica o nome, a unidade de engenharia e os limites superior e inferior de um canal de entrada analógica
GetAiSetup [1]	Informa o nome, a unidade de engenharia e os limites superior e inferior de um canal de entrada analógica
SetAoSetup [1]	Especifica o nome, a unidade de engenharia e os limites superior e inferior de um canal de saída analógica
GetAoSetup [1]	Informa o nome, a unidade de engenharia e os limites superior e inferior de um canal de saída analógica
SetCtrSetup [1]	Especifica o nome, a unidade de engenharia e o fator para conversão para unidade de engenharia de um canal de entrada de pulso

Função	Descrição
GetCtrSetup [1]	Informa o nome, a unidade de engenharia e o fator para conversão para unidade de engenharia de um canal de entrada de pulso
AiRange	Valor da Index-ésima faixa de entrada do conversor A/D. Valor positivo indica faixa unipolar. Por exemplo, o valor 10 indica uma faixa de entrada de 0 a 10. Valor negativo indica faixa bipolar. Por exemplo, o valor -10 indica uma faixa de entrada de -10 a 10.Obtém a configuração de canal A/D: nome, unidade de engenharia, limites da calibração, descrição e tipo de linearização:
AiRangeUnit	Unidade da Index-ésima faixa do conversor A/D
AoRange	Valor da Index-ésima faixa de saída do conversor D/A. Valor positivo indica faixa unipolar. Por exemplo, o valor 10 indica uma faixa de saída de 0 a 10. Valor negativo indica faixa bipolar. Por exemplo, o valor -10 indica uma faixa de saída de -10 a 10.
AoRangeUnit	Unidade da Index-ésima faixa de saída do conversor D/A
FResol	Valor da Index-ésima resolução na medição de frequência nos canais de contagem de pulso (em Hz).
TResol	Valor da Index-ésima resolução da medição de período nos canais de contagem de pulso (em microsegundos)
AiName [1, 2, 3]	Nome do sinal canal de entrada analógica
AiUnit [1, 2, 3]	Unidade de engenharia do sinal no canal de entrada analógica
AiHiLim [1, 2, 3]	Limite superior do sinal no canal de entrada analógica em unidade de engenharia
AiLoLim [1, 2, 3]	Limite inferior do sinal no canal de entrada analógica em unidade de engenharia
CtrName [1, 2, 3]	Nome do sinal no canal de entrada de pulso
CtrUnit [1, 2, 3]	Unidade de engenharia do sinal no canal de entrada de pulso
CtrFactor [1, 2, 3]	Fator de conversão do sinal no canal de contagem de pulso para unidade de engenharia. Valor em unidade de engenharia por pulso.
DiBitName [1, 2, 3]	Nome do bit de entrada digital
DiBitLow [1, 2, 3]	Valor para nível low de bit de entrada digital
DiBitHigh [1, 2, 3]	Valor para nível high de bit de entrada digital
AoName [1, 2, 3]	Nome do sinal no canal de saída analógica
AoUnit [1, 2, 3]	Unidade de engenharia do sinal no canal de saída analógica
AoHiLim [1, 2, 3]	Limite superior do canal de saída analógica em unidade de engenharia
AoLoLim [1, 2, 3]	Limite inferior do canal de saída analógica em unidade de engenharia
DoBitName [1, 2, 3]	Nome do bit de saída digital
DoBitLow [1, 2, 3]	Valor para nível low de bit de saída digital

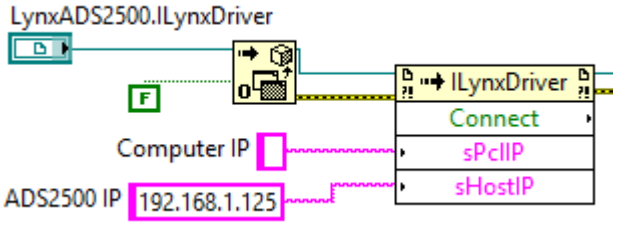
Notas:

1. Esses métodos foram incluídos na versão 2.0 da interface *ILynxDriver*.
2. Os valores das propriedades associadas a esses métodos são carregados do ADS2500 durante a chamada do método *Connect*. É mais fácil utilizar o assistente do *ADS2500* através do *Lynx@Net* para configurar os canais. Após a configuração ter sido salva no ADS2500, ela é carregada pelo driver através do método *Connect*.
3. A alteração dos valores das propriedades associadas a esses métodos só tem efeito durante a operação on line.

4.5. Método ILynxDriver.Connect

Após criação de uma a instância do automation *LynxADS2500.ILynxDriver*, deve-se estabelecer a conexão com o ADS2500. A conexão é realizada através da chamada do método *Connect* da interface *ILynxDriver*.

A figura abaixo ilustra um exemplo da chamada do método *ILynxDriver.Connect* disponível através do bloco *Invoke Node* da paleta *Connectivity / ActiveX* da janela *Functions* do LabVIEW.

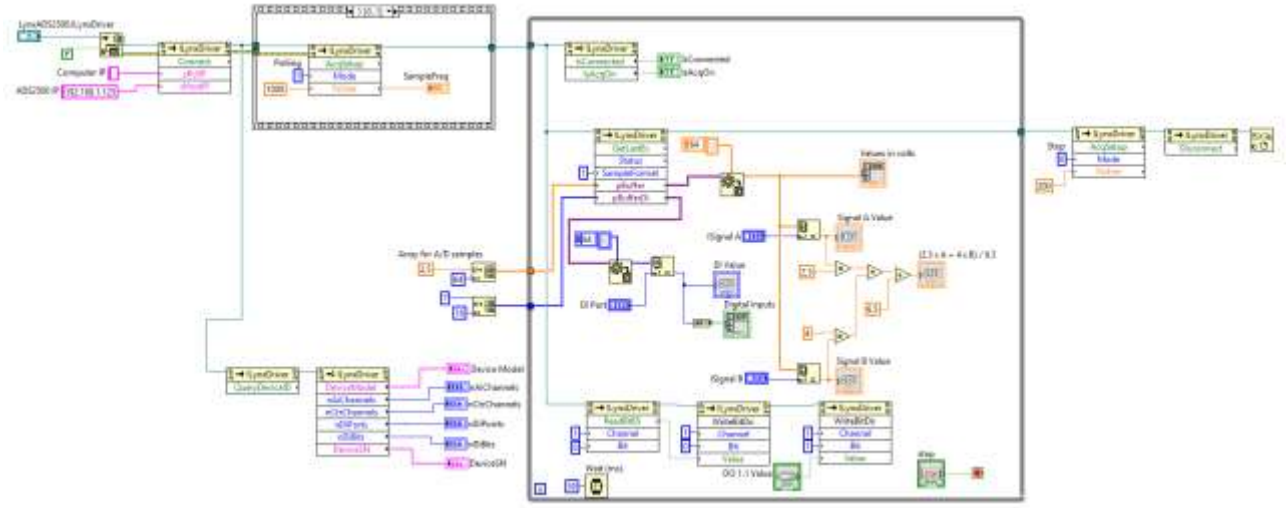


Terminal	Descrição
sPcIP	Terminal de entrada do tipo String. Especifique neste parâmetro o endereço IP da interface de rede do computador que será utilizada para conexão com o ADS2500. Este parâmetro pode ser deixado em branco. Nesse caso, o driver tentará se conectar com o ADS2500 através de uma das placas de rede do computador.
sHostIP	Terminal de entrada do tipo String. Especifique neste parâmetro o endereço IP do ADS2500. Este parâmetro pode ser deixado em branco. O endereço padrão do ADS2500 é `192.168.1.125`.
Valor retornado	Valor retornado do tipo boolean A função retorna *true* se a conexão com o ADS2500 foi estabelecida com sucesso ou *false* se a conexão não pode ser efetivada. Umas das causas da falha de conexão são: endereço IP errado, equipamento desligado, rota não estabelecida se o equipamento está em outra subrede, bloqueio do firewall e excedeu o número de conexões simultâneas do ADS2500.

4.6. Método ILynxDriver.Disconnect

Este método finaliza a conexão com o ADS2500.

4.7. Exemplo de Leitura no Modo Polling



A figura acima ilustra o diagrama de blocos do programa exemplo *Example_ADS2500_Display*. Esse exemplo realiza a aquisição de sinais no modo *polling* (*ILynxDriver.AcqSetup* com o parâmetro *Mode* igual a 1).

O modo *polling* realiza a coleta de todos os canais de entrada e é usualmente utilizado quando não há necessidade de registrar todas as amostras aquisitadas. O driver do ADS2500 realiza a aquisição de sinais com taxa de amostragem de 500 Hz e disponibiliza, para a aplicação, a última amostra aquisitada de cada canal de entrada. No caso dos canais de entrada analógica, é realiza uma média das últimas 25 amostras aquisitadas.

No loop principal do programa é realizada a chamada do método *ILynxDriver.GetLastEx* para a leitura da última amostra aquisitada de cada canal habilitado para aquisição.

A média ponderada entre dois canais selecionados é realizada e apresentada num display, assim como o valor desses canais.

A chamada do método *ILynxDriver.GetLastEx*, utilizado no exemplo, também poderia ser substituída por duas chamadas do método *ILynxDriver.ReadAiEx* para a leitura dos canais A/D A e B.

No loop também é efetuada a leitura do ponto de entrada digital 0. O valor dessa entrada é aplicada no ponto de saída digital 0. O ponto de saída digital 1 é atualizada através de um controle *Push Button*.

4.7.1. Método ILynxDriver.GetLastEx

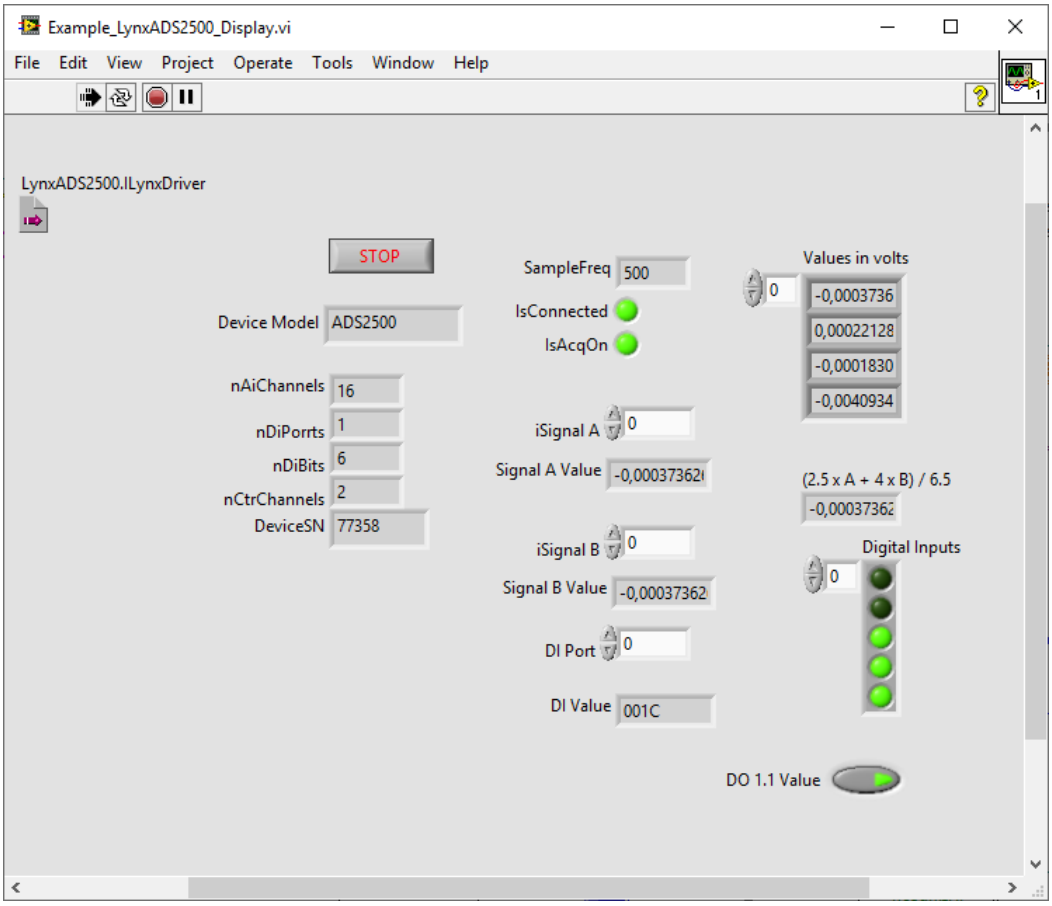
Este método retorna o valor da última amostra aquisitada de cada canal de entrada habilitado para aquisição.

As amostras dos sinais de entrada analógica e dos sinais de contagem são retornadas no formato especificado no parâmetro *SampleFormat*.

Terminal	Descrição
Status	Terminal de saída do tipo U16. Neste parâmetro é retornado o status da execução do método *GetLastEx*. Veja no enumerador *ieStatus*, no tópico 3.3, os valores possíveis retornados pelo método. O valor zero indica que não ocorreu nenhum erro.
SampleFormat	Especifica o formato com o qual as amostras de entrada analógicas e de contagem de pulso serão retornadas: • 0: normalizado (amostras normalizadas em +/- 1.0) • 1: unidade de entrada (volts para entrada analógica e pulsos para entrada de contagem de pulsos) • 2: unidade de engenharia
pBuffer	Terminal de entrada e saída do tipo Safearray. No terminal de entrada deve ser conectado um array do tipo ***double*** com tamanho maior ou igual ao número de sinais A/D e de contagem que estão sendo aquisitados pelo driver. Antes da chamada efetiva do método *GetLastEx*, o LabVIEW converte o array para *SafeArray*. O método *GetLastEx* devolve no SafeArray a última amostra aquisitada de cada canal A/D e de contagem que estão sendo aquisitados. Após a execução do método *GetLastEx*. o LabVIEW converte o SafeArray para um tipo *Variant*. Sendo assim, deve-se conectar no terminal de saída o bloco ***Variant to Data*** para converter o vetor para um array to tipo ***double*** do LabVIEW. A posição 0 do array corresponde à amostra do primeiro canal ativo, a posição 1 corresponde ao segundo canal ativo e assim sucessivamente. As primeiras posições do array correspondem às amostras dos sinais de entrada analógica que são seguidas das amostras dos sinais de entrada de contagem.
pBufferDI	Terminal de entrada e saída do tipo Safearray. No terminal de entrada deve ser conectado um array do tipo ***U32*** com tamanho maior ou igual ao número de ports de entrada digital que estão sendo aquisitados pelo provedor driver. Analogamente ao parâmetro *pBuffer*, deve-se conectar no terminal de saída o bloco ***Variant to Data*** para converter vetor para um array to tipo ***U32*** do LabVIEW.
Valor retornado	Valor retornado do tipo boolean A função retorna *true* se foi executada com sucesso. Se a função retornar *false* o valor retornado em *Status* deve ser consultado.

4.7.2. Painel do Exemplo

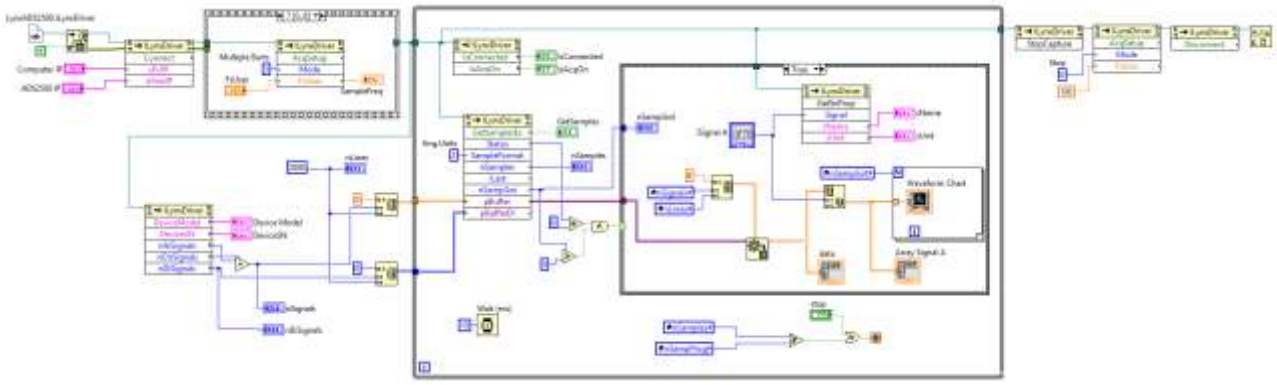
A figura abaixo apresenta a janela do painel do exemplo durante a execução.



4.8. Exemplo com Aquisição de Sinais

No exemplo anterior as amostras dos sinais são lidas através do método *ILynxDriver.GetLast*, que fornece a última amostra aquisitada de cada canal habilitado para aquisição.

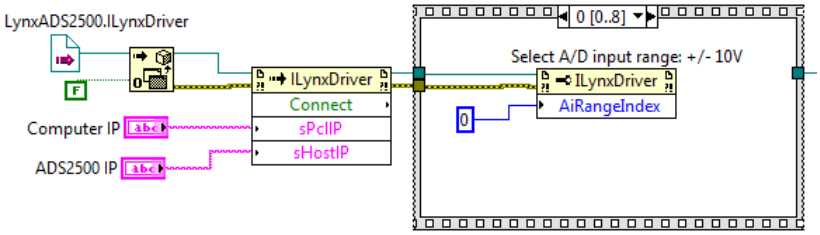
Quando for requerida a leitura das amostras continuamente numa dada taxa de amostragem, a programação em *LabVIEW* é um pouco diferente. A figura abaixo ilustra o diagrama de blocos do programa exemplo *TestADS2500_ScopeEx* para aquisição de sinais continua.



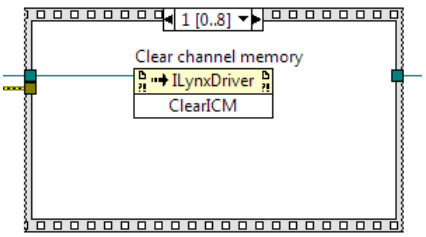
4.8.1. Programação da Aquisição de Sinais

No exemplo é utilizado o modo *Multiple Burts* para a aquisição de sinais. Nesse modo devem ser especificados os canais de entrada que serão aquisitados e a frequência de amostragem.

Após a chamada do método *ILynxDriver.Connect* para estabelecer a conexão com o ADS2500, segue o bloco *Stacked Sequence Structure* com a sequência para programação da aquisição de sinais. No primeiro bloco da sequência é selecionada a faixa de entrada do conversor A/D.

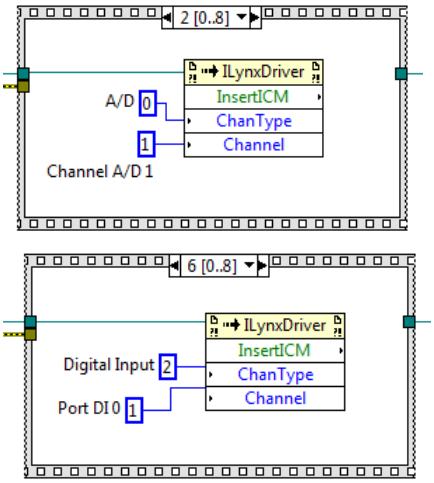


No segundo bloco da sequência é chamado o método *ILynxDriver.ClearICM* para zerar a memória de canais a serem aquisitados.



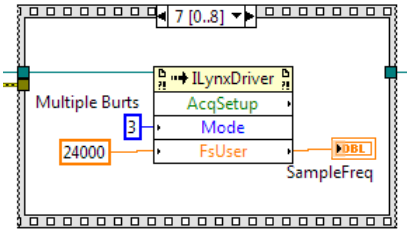
Nos blocos seguintes da sequência são incluídas chamadas do método *ILynxDriver.InsertICM* para informar quais canais de entrada devem ser aquisitados.

O método *ILynxDriver.InsertICM* recebe como parâmetros de entrada o tipo e o canal de entrada a ser aquisitado. A sequência deve ter uma chamada desse método para cada canal a ser aquisitado.

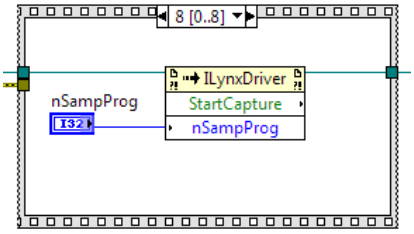


Após terem sido especificados os canais de entrada a serem aquisitados, é incluída a chamada do método *ILynxDriver.AcqSetup* para a definição do modo de aquisição de sinais e a frequência de amostragem. Na entrada *Mode* deve ser especificado o modo 3 (*Multiple Bursts*).

Na entrada *FsUser* deve ser informada a frequência de amostragem desejada. No exemplo está sendo utilizada a frequência de 24 kHz. Se a frequência de amostragem especificada não for suportada pelo ADS2500, o driver utiliza uma frequência de amostragem próxima da especificada. No terminal de saída *FsUser* é retornada a frequência de amostragem efetivamente programada.



No último bloco da sequência de programação da aquisição de sinais é chamado o método *ILynxDriver.StartCapture*. No terminal de entrada *nSampProg* deve ser passado o número de amostras a serem aquisitados por canal. No programa exemplo esse terminal é conectado a um controle para a especificação do número de amostras por canal que se deseja aquisitar.



4.8.2. Método ILynxDriver.AcqSetup

Este método é utilizado para a programação do modo da aquisição de sinais.

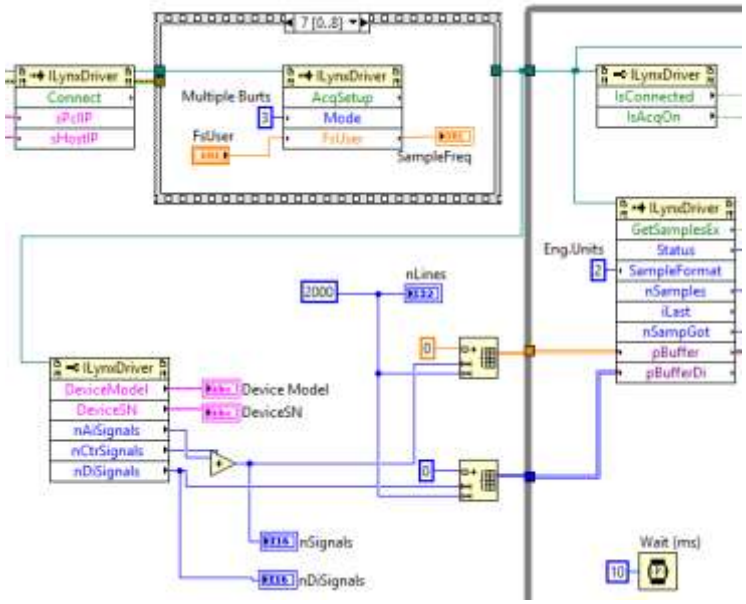
Terminal	Descrição
Mode	Terminal de entrada do tipo U16. Modo da aquisição de sinais: • 0: **Para a aquisição de sinais** Para a aquisição de sinais. • 1: **Inicia a aquisição no modo Polling** Nesse modo de aquisição a frequência de amostragem é definida pelo driver, usualmente 500 Hz, e apenas a última amostra (ou a média das últimas) de cada canal é disponibilizada para a aplicação. Ou seja, as amostras não são armazenadas num buffer circular para leitura periódica da aplicação através do método *GetSamples* ou *GetSamplesEx*. Todos os canais de entrada são aquisitados. • 2: **Inicia a aquisição no modo Single Burst** Nesse modo de aquisição os canais de entrada selecionados são amostrados na taxa de amostragem especificada e o equipamento é programado para enviar um *burst* dos sinais de cada vez. Este modo é apropriado nas aplicações em que os sinais de entrada são processados para gerar os sinais de saída a cada intervalo de amostragem. Devido ao tempo de processamento do *LabVIEW* em *Windows*, esse modo se aplica às aplicações com passo de loop maiores que 5 ms (200 Hz). • 3: **Inicia a aquisição no modo Multiple Burts** Esse modo de aquisição é análogo ao modo *Single Burst* com a diferença de que o equipamento é programado para enviar as amostras de vários *burts* de cada vez. Esse modo permite uma taxa de amostragem muito mais alta e se aplica aos casos em que não é necessária a atualização dos canais de saída ou a atualização das saídas não precisa ser na mesma taxa dos canais de entrada.
FsUser	Terminal de entrada e saída do tipo DBL. Frequência de amostragem dos sinais de entrada. No terminal de entrada deve ser passada a frequência de amostragem desejada. Se a frequência de amostragem especificada não for suportada pelo equipamento, a frequência mais próxima é utilizada pelo driver e retornada no terminal de saída
Valor retornado	Valor retornado do tipo boolean A função retorna *true* se a operação foi realizada com sucesso.

Se o modo de aquisição de sinais for *Single Burst* ou *Multiple Bursts*, o driver programa o equipamento para iniciar a aquisição de sinais. As amostras recebidas do equipamento pelo driver não são armazenadas no buffer circular do driver até que o método *StartCapture* seja executado. Apenas a última amostra de cada canal de entrada habilitado é armazenada pelo driver para ser lida pelos métodos *GetLast*, *GetLastEx*, *ReadAi*, *ReadAiEx*, *ReadCtr*, *ReadCtrEx*, *ReadDi, ReadBitDi* e *ReadBitDiEx*.

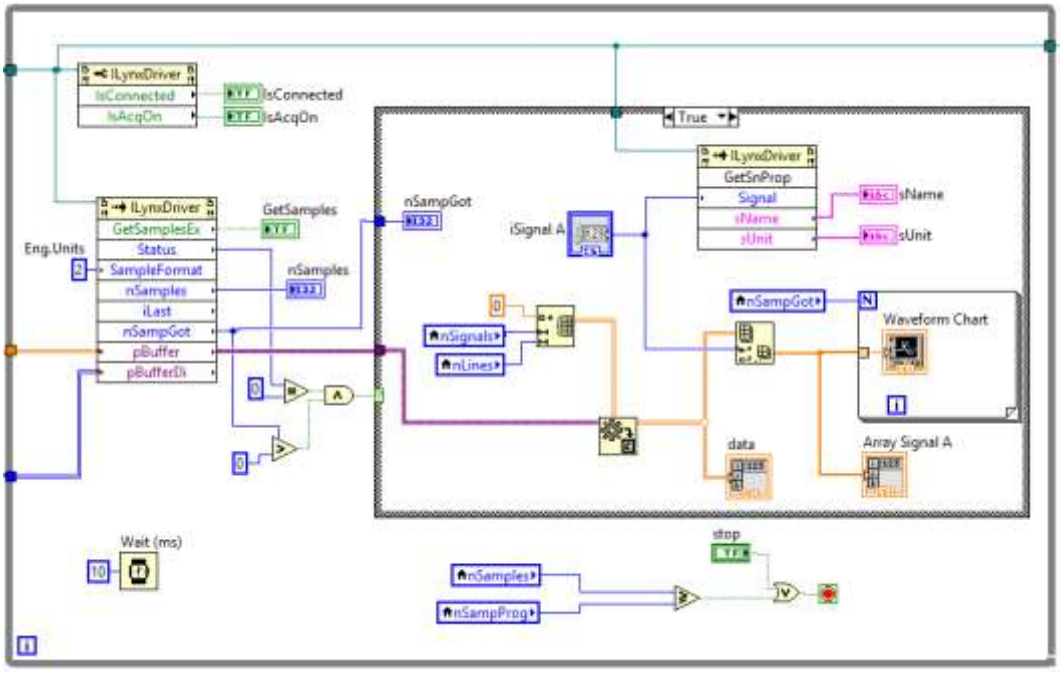
4.8.3. Arrays Utilizados na Chamada do ILynxDriver.GetSamplesEx

Antes do loop principal do programa exemplo devem ser definidas as dimensões dos arrays que serão utilizados para a leitura das amostras que estão sendo aquisitadas pelo driver. O primeiro array é do tipo *double* e é utilizado para a leitura das amostras dos canais de entrada analógica e dos canais de entrada de contagem de pulsos. O segundo array é do tipo U32 e é utilizado para a leitura das amostras das entradas digitais.

A figura seguinte ilustra como são criados esses arrays. O primeiro array é uma matriz onde a primeira dimensão é o número de canais de A/D e de contagem de pulso que foram habilitados para a aquisição. A segunda dimensão corresponde ao número máximo de amostras por canal que serão passados a cada chamada do método *ILynxDriver.GetSamplesEx*. O número de canais de entrada A/D e contagem de pulso habilitados para aquisição são obtidos através das propriedades *nAiSignals* e *nCtrSignals*. De modo análogo é criado o array para a leitura das amostras das entradas digitais.



A figura seguinte apresenta o loop principal do programa exemplo.



4.8.4. Método ILynxDriver.GetSamplesEx

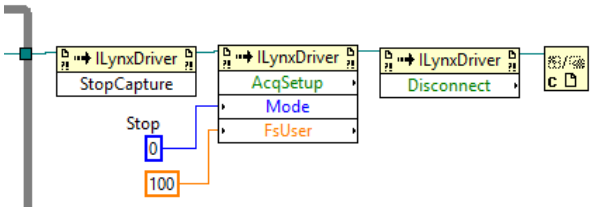
Este método retorna um conjunto de amostras aquisitadas desde a última chamada do método. O número de amostras retornadas por canal é limitado pelo tamanho dos safearrays.

As amostras dos sinais de entrada analógica e dos sinais de contagem são retornadas no formato especificado no parâmetro *SampleFormat*.

Terminal	Descrição
Status	Terminal de saída do tipo U16. Status da execução do método *GetSamplesEx*. Veja no enumerador *ieStatus*, no tópico 3.3, os valores possíveis retornados pelo método. O valor zero indica que não ocorreu nenhum erro.
SampleFormat	Especifica o formato com o qual as amostras de entrada analógicas e de contagem de pulso serão retornadas: • 0: normalizado (amostras normalizadas em +/- 1.0) • 1: unidade de entrada (volts para entrada analógica e pulsos para entrada de contagem de pulsos) • 2: unidade de engenharia
nSamples	Terminal de saída do tipo I32. Número de amostras aquisitadas por canal desde o início da aquisição de sinais.
iLast	Terminal de saída do tipo I32. Índice da última amostra retornada por esta chamada do método.
nSampGot	Terminal de saída do tipo I32. Índice da última amostra retornada por esta chamada do método.
pBuffer	Terminal de entrada e saída do tipo Safearray. No terminal de entrada deve ser conectado um array do tipo ***double*** de duas dimensões: Ns x Nt. Onde, Ns corresponde ao número de sinais ativos na aquisição (*nAiSignals* + *nCtrSignals*) e Nt corresponde ao número máximo de amostras por canal que podem ser retornadas na chamada do método *GetSamplesEx*. Antes da chamada efetiva do método *GetSamplesEx*, o LabVIEW converte o array para *SafeArray*. O método *GetSamplesEx* devolve no SafeArray um bloco de amostras correspondente a *nSampGot* amostras por canal. Após a execução do método *GetSamplesEx,* o LabVIEW converte o SafeArray para um tipo Variant. Sendo assim, deve-se conectar no terminal de saída o bloco ***Variant to Data*** para converter vetor para um array to tipo ***double*** de duas dimensões. Na matriz Ns x NT, as linhas correspondem aos sinais. As primeiras linhas correspondem às amostras dos sinais de entrada analógica e as linhas seguintes correspondem às amostras dos sinais de entrada de contagem.
pBufferDI	Terminal de entrada e saída do tipo Safearray. No terminal de entrada deve ser conectado um array do tipo ***U32*** de duas dimensões: Nd x NT. Onde, Nd corresponde ao número de ports de entrada digital ativos na aquisição (*nDiSignals*) e Nt corresponde ao número máximo de amostras por canal que podem ser retornadas na chamada do método *GetSamples*. Analogamente ao parâmetro *pBuffer*, deve-se conectar no terminal de saída o bloco ***Variant to Data*** para converter vetor para um array Nd x Nt to tipo ***U32*** d o LabVIEW.
Valor retornado	Valor retornado do tipo boolean A função retorna *true* se foi executada com sucesso. Se a função retornar *false* o valor retornado em *Status* deve ser consultado.

4.8.5. Finalização do Programa

O loop principal do programa é encerrado por comando de ***Stop*** do operador ou quando for atingido o número de amostras programado para a aquisição.

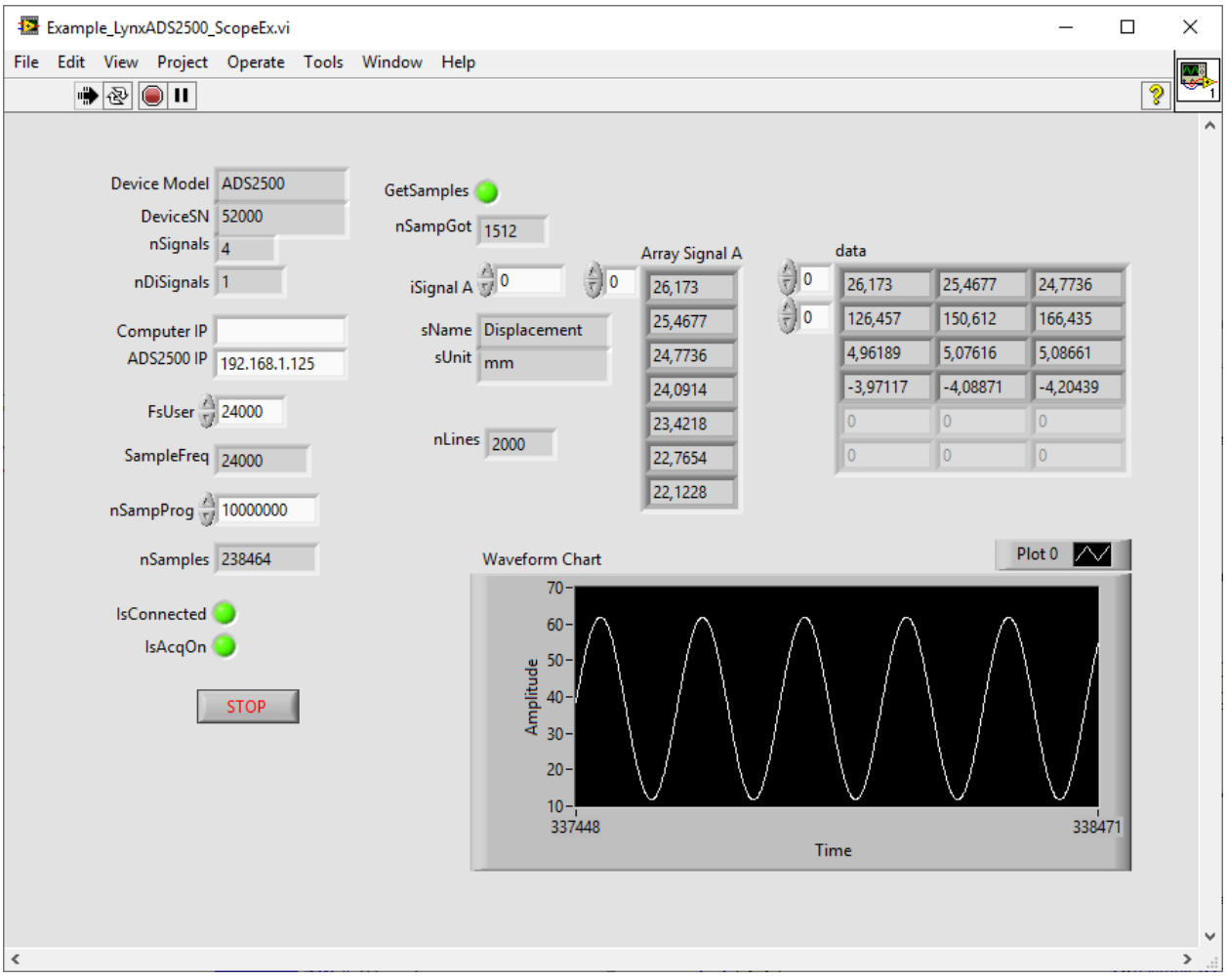


Após o loop principal segue uma sequência de blocos para a finalização do programa:

- Chamada do método *ILynxDriver.StopCapture* para a parada da captura e armazenamento das amostras dos sinais.
- Chamada do método *ILynxDriver.AcqSetup* com *Mode* igual 0 a para parar a aquisição de sinais.
- Chamada do método *ILynxDriver.Disconnect* para encerrar a conexão com o ADS2500.
- Bloco *Close Reference* para fechar a instância do driver do ADS2500.

4.8.6. Painel do Exemplo

A figura abaixo apresenta a janela do painel do programa.



5.1. Create COM Server

A instância do automation server do driver do ADS2500 é criada através do função *actxserver* do Matlab.

```
h = actxserver('LynxADS2500.LynxDriver');
```

Após criar a instância do *LynxADS2500.LynxDriver*, tem-se acesso às propriedades e aos métodos da interface *ILynxDriver* implementada pelo *automation LynxADS2500.LynxDriver*.

A variável ***h*** é o handle da instância do automation server criada pela chamada da função *actxserver*.

No decorrer desta documentação será utilizada a variável ***h*** como a instância do automation server *LynxADS2500.LynxDriver*.

5.2. Close Reference

Para finalizar a instância do servidor COM *LynxADS2500* deve ser executada a função delete.

```
h.delete;
```

5.3. Acesso às Propriedades da Interface ILynxDriver

Para acessar uma propriedade da interface *ILynxDriver*, basta referenciar o nome da propriedade através do handle da instância do automation server *LynxADS2500.LynxDriver*.

A tabela seguinte lista as propriedades da interface *ILynxDriver*.

Propriedade	Type	R/W	Descrição
LastErrorCode	`int16`	R	Código do último erro
IsConnected	`boolean`	R	Indica se está conectado com os equipamentos
sPcIP	`string`	R	Endereço IP da interface de rede do computador
sHostIP	`string`	R	Endereço IP do equipamento
DeviceModel	`string`	R	Modelo do hardware de aquisição de dados
DeviceSN	`string`	R	Número de série do hardware
DeviceTag	`string`	R	Tag do hardware
EnumStart1 [1]	`boolean`	R	Indica se a numeração dos canais se inicia em 1Tag do hardware
nAiChannels	`int16`	R	Número de canais A/D
nAiRange	`int16`	R	Número de opções de faixa de entrada do conversor A/D
AiRangeIndex	`uint16`	R/W	Seleciona a faixa de entrada do conversor A/D
nAoChannels	`int16`	R	Número de canais D/A
nAoRange	`int16`	R	Número de opções de faixa de faixa de saída do conversor D/A
AoRangeIndex	`uint16`	R/W	Seleciona a faixa de saída do conversor D/A
nCtrChannels	`int16`	R	Número de canais de contagem
nCtrBits	`int16`	R	Número de bits dos canais de contagem

Propriedade	Type	R/W	Descrição
nFResol	int16	R	Número de opções de resolução na medição de frequência nos canais de contagem de pulso.
FResolIndex	uint16	R/W	Seleciona a resolução da medição de frequência nos canais de contagem de pulso
nTResol	int16	R	Número de opções de resolução na medição de período nos canais de contagem de pulso.
TResolIndex	uint16	R/W	Seleciona a resolução da medição de período nos canais de contagem de pulso
nDiPorts	int16	R	Número de ports de entrada digital
nDiBits	int16	R	Número de bits por port entrada digital
nDoPorts	int16	R	Número de ports de saída digital
nDoBits	int16	R	Número de bits por port de saída digital
IsAcqOn	boolean	R	Indica se a aquisição de sinais está ativa
SampleFreq	double	R	Frequência de amostragem
nAiSignals	int16	R	Número de canais A/D habilitados para aquisição
nCtrSignals	int16	R	Número de canais de contagem habilitados para aquisição
nDiSignals	int16	R	Número de ports DI habilitados para aquisição
tSample	int64	R	Número de amostras aquisitadas desde o início da aquisição após a chamada do método *AcqSetup* com comando de partida da aquisição. Equivale ao contador de intervalos de amostragem e pode ser útil para determinar quando há um dado novo nas aplicações que utilizam o modo de aquisição *Single Burst*.
nSampProg	int32	R	Número de amostras a serem aquisitadas, programada no método *StartCapture*
nSamples	int32	R	Número de amostras aquisitadas por canal do total de *nSampProg*

Nota:

1. Propriedade incluída na versão 2.0 da interface *ILynxDriver*.

Por exemplo, para verificar se está estabelecida a conexão com o ADS2500 pode-se consultar a propriedades *IsConnected*.

```
r = h.IsConnected;
```

Para consultar o valor de todas as propriedades do servidor COM *LynxADS2500.LynxDriver* na linha de comando do Matlab, execute o seguinte comando:

```
h.get
```

5.4. Acesso aos Métodos da Interface ILynxDriver

Os métodos da interface com automation server *LynxADS2500.LynxDriver* são acessados através do handle da sua instância.

Por exemplo, na linha abaixo é chamado o método *Connect*.

```
r = h.Connect('', '192.168.1.125');
```

A tabela seguinte lista os métodos da interface *ILynxDriver*.

Função	Descrição
Connect	Estabelece a conexão o ADS2500.
Disconnect	Fecha a conexão com o ADS2500.
QueryDeviceID	Obtém identificação do equipamento
ProgCtr	Programa modo de operação de canal de contagem de pulso
ClearICM	Limpa a memória de canais
InsertICM	Insere canal na memória de canais
AcqSetup	Configura a aquisição de sinais
StartCapture	Inicia a captura da aquisição de sinais
StopCapture	Finaliza a captura da aquisição de sinais
GetSignalMap	Mapeamento dos sinais aquisitados com os canais físicos
GetSamples	Leitura das últimas amostras aquisitadas de cada sinal
GetSamplesEx [1]	Leitura das últimas amostras aquisitadas de cada sinal
GetLast	Leitura da última amostra aquisitada de cada sinal
GetLastEx [1]	Leitura da última amostra aquisitada de cada sinal
ReadAi	Leitura de canal de entrada analógica
ReadAiEx [1]	Leitura de canal de entrada analógica
ReadCtr	Leitura de canal de contagem de pulso
ReadCtrEx [1]	Leitura de canal de contagem de pulso
ReadDi	Leitura de port de entrada digital
ReadBitDi	Leitura de bit de port de entrada digital
ReadBitDiEx [1]	Leitura de bit de port de entrada digital
ReadDo	Leitura de port de saída digital (última escrita)
ReadBitDo	Leitura de bit de port de saída digital
ReadBitDoEx [1]	Leitura de bit de port de saída digital
WriteDo	Escrita em port de saída digital
WriteAo	Escrita em canal de saída analógica
WriteAoEx [1]	Escrita em canal de saída analógica
GetSnProp [1]	Informa o nome e a unidade de engenharia de um sinal habilitado para aquisição de sinais (canal A/D ou de contagem de pulso)
SetAiSetup [1]	Especifica o nome, a unidade de engenharia e os limites superior e inferior de um canal de entrada analógica
GetAiSetup [1]	Informa o nome, a unidade de engenharia e os limites superior e inferior de um canal de entrada analógica
SetAoSetup [1]	Especifica o nome, a unidade de engenharia e os limites superior e inferior de um canal de saída analógica
GetAoSetup [1]	Informa o nome, a unidade de engenharia e os limites superior e inferior de um canal de saída analógica
SetCtrSetup [1]	Especifica o nome, a unidade de engenharia e o fator para conversão para unidade de engenharia de um canal de entrada de pulso

Função	Descrição
GetCtrSetup [1]	Informa o nome, a unidade de engenharia e o fator para conversão para unidade de engenharia de um canal de entrada de pulso
AiRange	Valor da Index-ésima faixa de entrada do conversor A/D. Valor positivo indica faixa unipolar. Por exemplo, o valor 10 indica uma faixa de entrada de 0 a 10. Valor negativo indica faixa bipolar. Por exemplo, o valor -10 indica uma faixa de entrada de -10 a 10.Obtém a configuração de canal A/D: nome, unidade de engenharia, limites da calibração, descrição e tipo de linearização:
AiRangeUnit	Unidade da Index-ésima faixa do conversor A/D
AoRange	Valor da Index-ésima faixa de saída do conversor D/A. Valor positivo indica faixa unipolar. Por exemplo, o valor 10 indica uma faixa de saída de 0 a 10. Valor negativo indica faixa bipolar. Por exemplo, o valor -10 indica uma faixa de saída de -10 a 10.
AoRangeUnit	Unidade da Index-ésima faixa de saída do conversor D/A
FResol	Valor da Index-ésima resolução na medição de frequência nos canais de contagem de pulso (em Hz).
TResol	Valor da Index-ésima resolução da medição de período nos canais de contagem de pulso (em microsegundos)
AiName [1, 2, 3]	Nome do sinal canal de entrada analógica
AiUnit [1, 2, 3]	Unidade de engenharia do sinal no canal de entrada analógica
AiHiLim [1, 2, 3]	Limite superior do sinal no canal de entrada analógica em unidade de engenharia
AiLoLim [1, 2, 3]	Limite inferior do sinal no canal de entrada analógica em unidade de engenharia
CtrName [1, 2, 3]	Nome do sinal no canal de entrada de pulso
CtrUnit [1, 2, 3]	Unidade de engenharia do sinal no canal de entrada de pulso
CtrFactor [1, 2, 3]	Fator de conversão do sinal no canal de contagem de pulso para unidade de engenharia. Valor em unidade de engenharia por pulso.
DiBitName [1, 2, 3]	Nome do bit de entrada digital
DiBitLow [1, 2, 3]	Valor para nível low de bit de entrada digital
DiBitHigh [1, 2, 3]	Valor para nível high de bit de entrada digital
AoName [1, 2, 3]	Nome do sinal no canal de saída analógica
AoUnit [1, 2, 3]	Unidade de engenharia do sinal no canal de saída analógica
AoHiLim [1, 2, 3]	Limite superior do canal de saída analógica em unidade de engenharia
AoLoLim [1, 2, 3]	Limite inferior do canal de saída analógica em unidade de engenharia
DoBitName [1, 2, 3]	Nome do bit de saída digital
DoBitLow [1, 2, 3]	Valor para nível low de bit de saída digital
DoBitHigh [1, 2, 3]	Valor para nível high de bit de saída digital
DoBitHigh [1, 2, 3]	Valor para nível high de bit de saída digital

Notas:

1. Esses métodos foram incluídos na versão 2.0 da interface *ILynxDriver*.
2. Esses métodos são propriedades da interface *ILynxDriver* que o Matlab interpretada como métodos. No entanto, o acesso é apenas para leitura. Para alterar os valores dessas propriedades, utilize os métodos *SetAiSetup*, *SetAoSetup, SetCtrSetup*, *SetDiSetup* e *SetDoSetup*.
3. Os valores das propriedades associadas a esses métodos são carregados do ADS2500 durante a chamada do método *Connect*. É mais fácil utilizar o assistente do *ADS2500* através do *Lynx@Net* para configurar os canais. Após a configuração ter sido salva no ADS2500, ela é carregada pelo driver através do método *Connect*.

Para consultar a lista de métodos servidor COM LynxADS2500.LynxDriver na linha de comando do Matlab, execute o seguinte comando:

```
h.methods('-full')
```

5.5. Método ILynxDriver.Connect

Após criação de uma a instância do automation *LynxADS2500.LynxDriver*, deve-se estabelecer a conexão com o ADS2500. A conexão é realizada através da chamada do método *Connect* da interface *ILynxDriver*.

```
r = h.Connect(sPcIP, sHostIP);
```

Parâmetro	Descrição
sPcIP	Parâmetro de entrada do tipo String. Especifique neste parâmetro o endereço IP da interface de rede do computador que será utilizada para conexão com o ADS2500. Este parâmetro pode ser deixado em branco. Nesse caso, o driver tentará se conectar com o ADS2500 através de uma das placas de rede do computador.
sHostIP	Parâmetro de entrada do tipo String. Especifique neste parâmetro o endereço IP do ADS2500. Este parâmetro pode ser deixado em branco. O endereço padrão do ADS2500 é `192.168.1.125`.
r	Valor retornado do tipo boolean A função retorna *true* se a conexão com o ADS2500 foi estabelecida com sucesso ou *false* se a conexão não pode ser efetivada. Umas das causas da falha de conexão são: endereço IP errado, equipamento desligado, rota não estabelecida se o equipamento está em outra subrede, bloqueio do firewall e excedeu o número de conexões simultâneas do ADS2500.

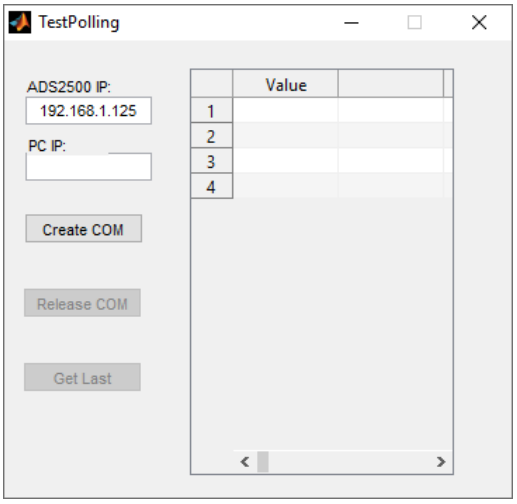
5.6. Método ILynxDriver.Disconnect

Este método finaliza a conexão com o ADS2500.

```
h.Disconnect;
```

5.7. Exemplo de Leitura do Modo Polling

A arquivo ***TestPolling.m*** ilustra um exemplo em Matlab para aquisição de sinais no modo polling. A interface gráfica desse exemplo está contida no arquivo ***TestPolling.fig*** ilustrado na *figura* abaixo.



O programa exemplo possui os seguintes controles.

- Botão ***Create COM***.
  No tratamento desse botão é criada uma instância do automation server *LynxADS2500.LynxDriver* através da função ***actxserver*** do Matlab. Esse seguida é solicitada a conexão de aquisição de sinais com o ADS2500. Para esse fim é executado o método ***Connect*** do automation server. Os parâmetros utilizados nessa chamada são os especificados nos controles *ADS2500 IP* e *PC IP*. A aquisição de sinais é programada no modo *Polling* (parâmetro *Mode* igual a 1 na chamada do método *AcqSetup*).

```
handles.h = actxserver('LynxADS2500.LynxDriver');
sPcIP  = get(handles.edPcIP, 'String');
sDevIP = get(handles.edDevIP, 'String');
handles.fOpened = handles.h.Connect (sPcIP, sDevIP);
handles.h.AiRangeIndex = 0;   % Select +/- 10.0V
handles.h.AcqSetup (1, 100);
```

- **Tabela de valores das amostras**
  O controle *uiTable* do tipo *Table* foi acrescentado na janela do exemplo para a apresentação da leitura das amostras dos sinais aquisitados pelo driver.
- Botão ***Get Last***.
  Chamada do método *ILynxDriver.GetLastEx* para a leitura da última amostra aquisitada de cada canal habilitado para aquisição de sinais. Na chamada do *GetLastEx* é especificado a leitura em unidade de entrada (volts).

```
[r, status, handles.d, handles.b] = handles.h.GetLastEx(1,
handles.d, handles.b);
set(handles.uiTable, 'Data', handles.d');
```

- Botão ***Release COM***
  No tratamento deste botão é finalizada a conexão com o ADS2500 e é fechada a instância do automation server *LynxADS2500.LynxDriver*. Basicamente são executados os seguintes comandos no tratamento desse botão.

```
handles.h.AcqSetup (0, 100);
handles.h.Disconnect;
handles.h.delete;
```

5.7.1. Método ILynxDriver.GetLastEx

Este método retorna o valor da última amostra aquisitada de cada sinal habilitado para aquisição de sinais.

```
[r, Status, d, b] = h.GetLastEx (sf, d, b);
```

Parâmetro	Descrição
d	Parâmetro de entrada e saída do tipo Safearray. No parâmetro ***d*** deve ser passado um vetor do tipo ***double*** com tamanho maior ou igual ao número de sinais A/D e de contagem que estão sendo aquisitados pelo driver O método *GetLastEx* devolve no vetor a última amostra aquisitada de cada canal A/D e de contagem que estão sendo aquisitados. A posição 1 do vetor corresponde à amostra do primeiro canal ativo, a posição 2 corresponde ao segundo canal ativo e assim sucessivamente. As primeiras posições do array correspondem às amostras dos sinais de entrada analógica que são seguidas das amostras dos sinais de entrada de contagem.
sf	Parâmetro de entrada. Especifica o formato com o qual as amostras de entrada analógicas e de contagem de pulso serão retornadas: • 0: normalizado (amostras normalizadas em +/- 1.0) • 1: unidade de entrada (volts para entrada analógica e pulsos para entrada de contagem de pulsos) • 2: unidade de engenharia
b	Parâmetro de entrada e saída do tipo Safearray. No parâmetro ***b*** deve ser conectado um array do tipo ***uint32*** com tamanho maior ou igual ao número de ports de entrada digital que estão sendo aquisitados pelo driver.
Status	Parâmetro de saída do tipo uint16. Neste parâmetro é retornado o status da execução do método *GetLastEx*. Veja no enumerador *ieStatus*, no tópico 3.3, os valores possíveis retornados pelo método. O valor zero indica que não ocorreu nenhum erro.
r	Parâmetro de saída do tipo boolean A função retorna *true* se foi executada com sucesso. Se a função retornar *false* o valor retornado em *Status* deve ser consultado.

5.7.2. Arquivo TestPolling.m

```
function varargout = TestPolling(varargin)
% TESTPOLLING MATLAB code for TestPolling.fig
%      TESTPOLLING, by itself, creates a new TESTPOLLING or raises the existing
%      singleton*.
%
%      H = TESTPOLLING returns the handle to a new TESTPOLLING or the handle to
%      the existing singleton*.
%
%      TESTPOLLING('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TESTPOLLING.M with the given input arguments.
%
%      TESTPOLLING('Property','Value',...) creates a new TESTPOLLING or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before TestPolling_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to TestPolling_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
```

```
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help TestPolling

% Last Modified by GUIDE v2.5 24-Mar-2015 09:57:12

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @TestPolling_OpeningFcn, ...
                   'gui_OutputFcn',  @TestPolling_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before TestPolling is made visible.
function TestPolling_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to TestPolling (see VARARGIN)

% Choose default command line output for TestPolling
handles.output = hObject;

% cria matriz para o GetLast e inicia flags
handles.d=zeros(1, 20, 'double');
handles.b=zeros(1, 2, 'uint32');
handles.fOpened = false;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes TestPolling wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Executes just before TestPolling is made visible.
function update_display(obj, event, hObject)
handles = guidata(hObject);
if handles.fAcqOn
    [r, status, handles.d, handles.b] = handles.h.GetLastEx(1, handles.d, handles.b);
    set(handles.uiTable, 'Data', handles.d');
end


% --- Outputs from this function are returned to the command line.
function varargout = TestPolling_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in btCreateCOM.
function btCreateCOM_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to btCreateCOM (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.h = actxserver('LynxADS2500.LynxDriver');
set (handles.lblWarning, 'Visible', 'On');
guidata(hObject, handles);
sPcIP  = get(handles.edPcIP, 'String');
sDevIP = get(handles.edDevIP, 'String');
handles.fOpened = handles.h.Connect (sPcIP, sDevIP);
set (handles.lblWarning, 'Visible', 'Off');
if handles.fOpened
    handles.h.AiRangeIndex = 0;   % Select +/- 10.0V
    handles.h.AcqSetup (1, 100);
    set (handles.btCreateCOM, 'Enable', 'Off');
    set (handles.btReleaseCOM, 'Enable', 'On');
    set (handles.btGetLast, 'Enable', 'On');
else
    handles.h.delete;
end
guidata(hObject, handles);


% --- Executes on button press in btReleaseCOM.
function btReleaseCOM_Callback(hObject, eventdata, handles)
% hObject    handle to btReleaseCOM (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if handles.fOpened
    handles.h.AcqSetup (0, 100);
    handles.h.Disconnect;
    handles.h.delete;
    handles.fOpened = false;
end
set (handles.btCreateCOM, 'Enable', 'On');
set (handles.btReleaseCOM, 'Enable', 'Off');
set (handles.btGetLast, 'Enable', 'Off');
guidata(hObject, handles);


% --- Executes on button press in btGetLast.
function btGetLast_Callback(hObject, eventdata, handles)
% hObject    handle to btGetLast (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if handles.fOpened
    % read signal in engineering units
    [r, status, handles.d, handles.b] = handles.h.GetLastEx(2, handles.d, handles.b);
    set(handles.uiTable, 'Data', handles.d');
end


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if handles.fOpened
    handles.h.Disconnect;
    handles.h.delete;
    handles.fOpened = false;
end
% Hint: delete(hObject) closes the figure
delete(hObject);


% --- Executes during object creation, after setting all properties.
function figure1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

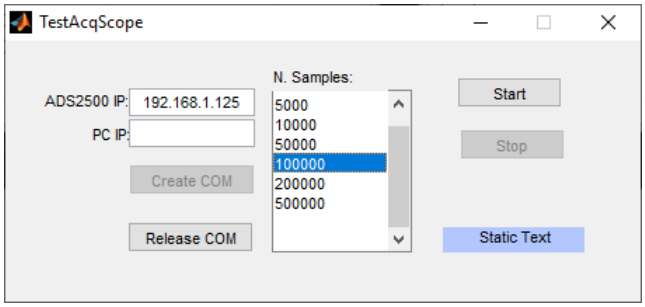
```
function edDevIP_Callback(hObject, eventdata, handles)
% hObject    handle to edDevIP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edDevIP as text
%        str2double(get(hObject,'String')) returns contents of edDevIP as a double


% --- Executes during object creation, after setting all properties.
function edDevIP_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edDevIP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edPcIP_Callback(hObject, eventdata, handles)
% hObject    handle to edPcIP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edPcIP as text
%        str2double(get(hObject,'String')) returns contents of edPcIP as a double


% --- Executes during object creation, after setting all properties.
function edPcIP_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edPcIP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

5.8. Exemplo com Aquisição de Dados

No exemplo anterior as amostras dos sinais são lidas através do método *ILynxDriver.GetLastEx*, que fornece a última amostra aquisitada de cada canal habilitado para aquisição.

Quando for requerida a leitura das amostras continuamente numa dada taxa de amostragem, a programação em *Matlab* é um pouco diferente.

O arquivo ***TestAcqScope.m*** é um exemplo de aquisição de sinais continua com armazenagem das amostras. A interface gráfica com o usuário desse exemplo é definida no arquivo ***TestAcqScope.fig*** e ilustrada na figura abaixo.



O programa exemplo possui os seguintes controles.

- Botão ***Create COM***
  No tratamento desse botão é criada uma instância do automation server *LynxADS2500.LynxDriver* através da função ***actxserver*** do Matlab. Esse seguida é solicitada a conexão de aquisição de sinais com o ADS2500. Para esse fim é executado o método ***Connect*** do automation server. Os parâmetros utilizados nessa chamada são os especificados nos controles *ADS2500 IP* e *PC IP*.

```
handles.h = actxserver('LynxADS2500.LynxDriver');
sPcIP  = get(handles.edPcIP, 'String');
sDevIP = get(handles.edDevIP, 'String');
handles.fOpened = handles.h.Connect (sPcIP, sDevIP);
```

- Botão ***Start***
  No tratamento desse botão é programado e iniciada a aquisição de sinais no modo *Multiple Burts* numa taxa de amostragem de 8 kHz. É habilita a aquisição dos canais A/D 1, 2, 3 e 5 e o canal de contagem de pulso 1. O código fonte em Matlab para o tratamento desse botão pode ser consultado no arquivo *TestAcqScope.m*.

```
handles.h.AiRangeIndex = 0;
handles.h.ClearICM;
handles.h.InsertICM (0, 1);
handles.h.InsertICM (0, 2);
handles.h.InsertICM (0, 3);
handles.h.InsertICM (0, 5);
handles.h.InsertICM (1, 1);
handles.Fs = handles.h.AcqSetup (3, 8000);
iNS = get(handles.lbNSamples,'Value');
handles.nSampProg = handles.TabNS(iNS);
handles.h.StartCapture (handles.nSampProg)
```

- Botão ***Stop***
  Esse botão permite parar a aquisição de sinais para o programa em Matlab antes de ser aquisitado o número de amostras previamente programado. Basicamente é executado o seguinte código no tratamento desse botão:

```
handles.h.StopCapture;
```

```
handles.h.AcqSetup(0, 100);
```

- ***Timer***
  O programa define um timer para realizar chamadas periódicas do método *GetSamplesEx* e para a atualização da apresentação gráfica do sinal aquisitado. O formato de leitura especificado na chamada do *GetSamplesEx* é em unidade de engenharia.

- Botão ***Release COM***
  No tratamento deste botão é finalizada a conexão com o ADS2500 e é fechada a instância do automation server *LynxADS2500.LynxDriver*. Basicamente são executados os seguintes comandos no tratamento desse botão.

```
handles.h.AcqSetup (0, 100);
handles.h.Disconnect;
handles.h.delete;
```

5.8.1. Método ILynxDriver.AcqSetup

Este método é utilizado para a programação do modo da aquisição de sinais.

```
[r, FsReal] = h.AcqSetup (Mode, FsUser);
```

Parâmetro	Descrição
Mode	Parâmetro de entrada do tipo unint16. Modo da aquisição de sinais: • 0: **Para a aquisição de sinais** Para a aquisição de sinais. • 1: **Inicia a aquisição no modo Polling** Nesse modo de aquisição a frequência de amostragem é definida pelo driver, usualmente 500 Hz, e apenas a última amostra (ou a média das últimas) de cada canal é disponibilizada para a aplicação. Ou seja, as amostras não são armazenadas num buffer circular para leitura periódica da aplicação através do método *GetSamples*. Todos os canais de entrada são aquisitados. • 2: **Inicia a aquisição no modo Single Burst** Nesse modo de aquisição os canais de entrada selecionados são amostrados na taxa de amostragem especificada e o equipamento é programado para enviar um *burst* dos sinais de cada vez. Este modo é apropriado nas aplicações em que os sinais de entrada são processados para gerar os sinais de saída a cada intervalo de amostragem. Devido ao tempo de processamento do *Matlab* em *Windows*, esse modo se aplica às aplicações com passo de loop maiores que 5 ms (200 Hz). • 3: **Inicia a aquisição no modo Multiple Burts** Esse modo de aquisição é análogo ao modo *Single Burst* com a diferença de que o equipamento é programado para enviar as amostras de vários *burts* de cada vez. Esse modo permite uma taxa de amostragem muito mais alta e se aplica aos casos em que não é necessária a atualização dos canais de saída ou a atualização das saídas não precisa ser na mesma taxa dos canais de entrada.
FsUser	Parâmetro de entrada tipo double. Frequência de amostragem dos sinais de entrada. Se a frequência de amostragem especificada não for suportada pelo equipamento, a frequência mais próxima é utilizada pelo driver
FsReal	Parâmetro de saída tipo double. Frequência de amostragem utilizada pelo driver.
r	Valor retornado do tipo boolean A função retorna *true* se a operação foi realizada com sucesso.

Se o modo de aquisição de sinais for *Single Burst* ou *Multiple Bursts*, o driver programa o equipamento para iniciar a aquisição de sinais. As amostras recebidas do equipamento pelo driver não são armazenadas no buffer circular do driver até que o método *StartCapture* seja executado. Apenas a última amostra de cada canal de entrada habilitado é armazenada pelo driver para ser lida pelos métodos *GetLast*, *GetLastEx*, *ReadAi*, *ReadAiEx*, *ReadCtr*, *ReadCtrEx*, *ReadDi*, *ReadBitDi* e *ReadBitDiEx*.

5.8.2. Método ILynxDriver.StartCapture

O início da aquisição de sinais propriamente dita é comanda pelo método *AcqSetup*. O driver passa a receber os pacotes de dados do ADS2500. No entanto, as amostras recebidas não são armazenadas no buffer circular pelo driver. Somente a última amostra de cada canal habilitado é salvo para ser lido pelo método *GetLast* ou *GetLastEx*. O método *StartCapture* sinaliza o driver para passar a armazenar as amostras recebidas até atingir o número de amostras por canal programado no parâmetro *nSampProg*. O software aplicativo deve chamar o método *GetSamples* ou *GetSamplesEx* periodicamente para que não ocorra overrun no buffer circular do driver.

```
[r] = h.StartCapture (nSampProg);
```

Parâmetro	Descrição
nSampProg	Parâmetro de entrada tipo int32. Número de amostras por canal a serem aquisitadas.
r	Valor retornado do tipo boolean A função retorna *true* se a operação foi realizada com sucesso.

5.8.3. Método ILynxDriver.StopCapture

Este método finaliza a captura das amostras para o buffer circular do driver.

5.8.4. Método ILynxDriver.GetSamplesEx

Este método retorna um conjunto de amostras aquisitadas desde a última chamada do método ou que ainda não foram lidas pela aplicação. O número de amostras retornadas por canal é limitado pelo tamanho dos safearrays.

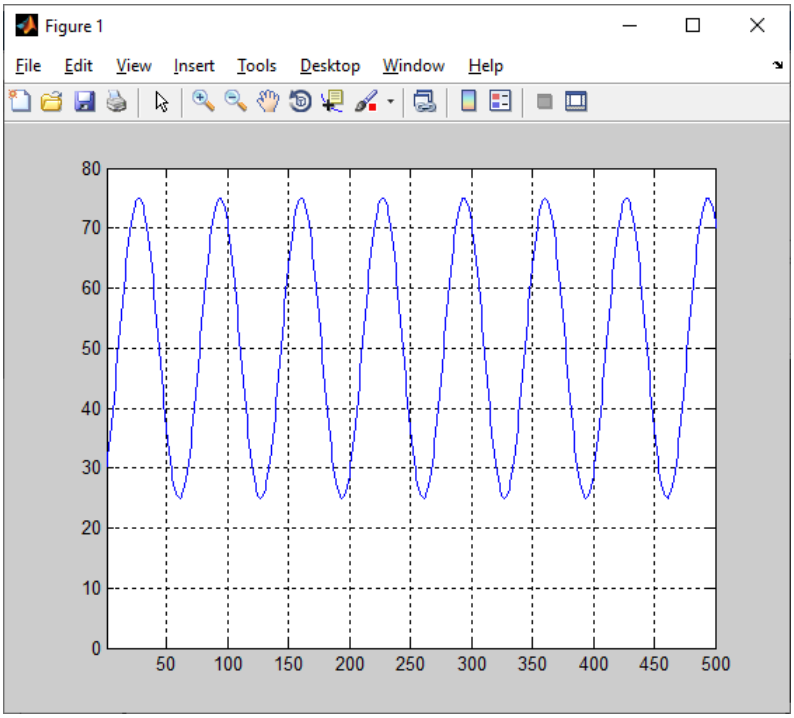
Este método é utilizado em conjunto com o método *StartCapture*. Na chamada do *StartCapture* é passado o número de amostras a serem aquisitadas por canal. As amostras aquisitadas são armazenadas num buffer circular e devem ser lidas periodicamente através do *GetSamplesEx*.

```
  [r, Status, nSamples, iLast, nSampGot, md, mb] = h.GetSamplesEx(sf, md,
mb);
```

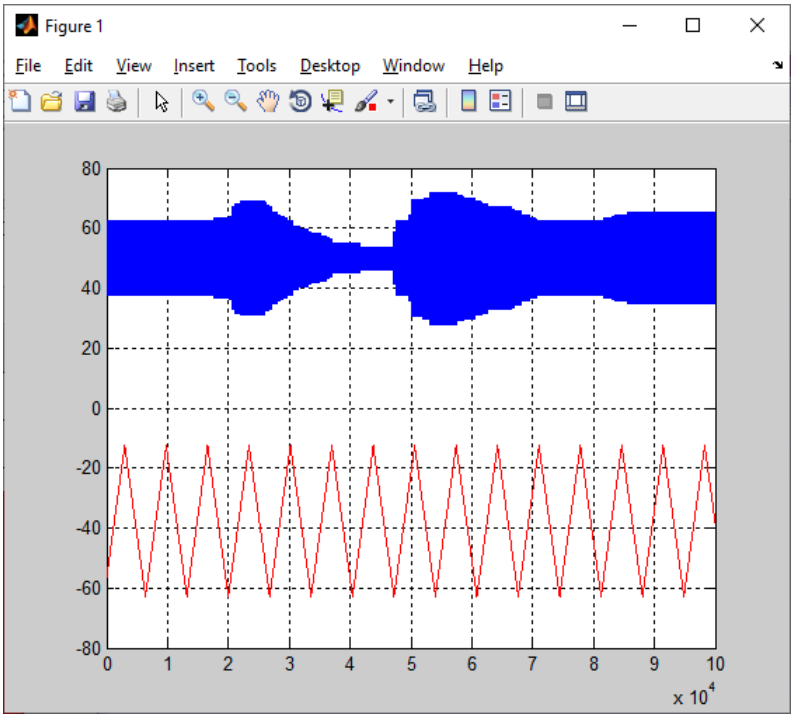
Parâmetro	Descrição
md	Parâmetro de entrada e saída do tipo Safearray. No parâmetro ***md*** deve ser passada uma matriz do tipo ***double*** de duas dimensões: Ns x Nt. Onde, Ns corresponde ao número de sinais ativos na aquisição (*nAiSignals* + *nCtrSignals*) e Nt corresponde ao número máximo de amostras por canal que podem ser retornadas na chamada do método *GetSamplesEx*.
sf	Parâmetro de entrada. Especifica o formato com o qual as amostras de entrada analógicas e de contagem de pulso serão retornadas: • 0: normalizado (amostras normalizadas em +/- 1.0) • 1: unidade de entrada (volts para entrada analógica e pulsos para entrada de contagem de pulsos) • 2: unidade de engenharia
mb	Parâmetro de entrada e saída do tipo Safearray. No parâmetro ***mb*** deve ser passada uma matriz do tipo ***uint32*** de duas dimensões: Nd x NT. Onde, Nd corresponde ao número de ports de entrada digital ativos na aquisição (*nDiSignals*) e Nt corresponde ao número máximo de amostras por canal que podem ser retornadas na chamada do método *GetSamplesEx*.
Status	Parâmetro de saída do tipo uint16. Status da execução do método *GetSamplesEx*. Veja no enumerador *ieStatus*, no tópico 3.3, os valores possíveis retornados pelo método. O valor zero indica que não ocorreu nenhum erro.
nSamples	Parâmetro de saída do tipo int32. Número de amostras aquisitadas por canal desde o início da aquisição de sinais.
iLast	Parâmetro de saída do tipo int32. Índice da última amostra retornada por esta chamada do método.
nSampGot	Parâmetro de saída do tipo int32. Índice da última amostra retornada por esta chamada do método.
r	Parâmetro de saída do tipo boolean A função retorna *true* se foi executada com sucesso. Se a função retornar *false* o valor retornado em *Status* deve ser consultado.

5.8.5. Scope para Visualização do Sinal

A título de exemplo, o canal A/D 1 é apresentado durante a aquisição de sinais num *scope* com as últimas 500 amostras aquisitadas até então. Para atualização do scope é utilizado um timer que roda a cada 50 ms.



No final da aquisição de sinais o *scope* é atualizado com todas as amostras programadas para a aquisição. No *scope* são apresentados os gráficos sobrepostos dos dois primeiros canais habilitados.



5.8.6. Arquivo TestAcqScope.m

```
function varargout = TestAcqScope(varargin)
% TESTACQSCOPE MATLAB code for TestAcqScope.fig
%      TESTACQSCOPE, by itself, creates a new TESTACQSCOPE or raises the existing
%      singleton*.
%
%      H = TESTACQSCOPE returns the handle to a new TESTACQSCOPE or the handle to
%      the existing singleton*.
%
%      TESTACQSCOPE('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in TESTACQSCOPE.M with the given input arguments.
%
%      TESTACQSCOPE('Property','Value',...) creates a new TESTACQSCOPE or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before TestAcqScope_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to TestAcqScope_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help TestAcqScope

% Last Modified by GUIDE v2.5 24-Mar-2015 09:41:02

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @TestAcqScope_OpeningFcn, ...
                   'gui_OutputFcn',  @TestAcqScope_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before TestAcqScope is made visible.
function TestAcqScope_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to TestAcqScope (see VARARGIN)

% Choose default command line output for TestAcqScope
handles.output = hObject;

% cria matriz para o GetSamples
handles.BufN = zeros(18, 16384, 'double');
handles.BufDi = zeros(8, 16384, 'uint32');

% cria o buffer para armazenar as amostras: 18 x 8192
handles.B = zeros(18, 8192, 'double');
handles.n = 0;
handles.Fs = 2000;
handles.fOpened = false;
handles.fAcqOn = false;
handles.fTimer = false;
```

```
handles.hPlot  = figure;
set(handles.hPlot, 'Visible', 'off');
% cria o timer para atualização da leitura
handles.tmr = timer(...
    'ExecutionMode', 'fixedRate', ...
    'Period', 0.05, ...
    'TimerFcn', {@update_samples, hObject});

% opcoes de número de amostras
handles.TabNS = [1000, 5000, 10000, 50000, 100000, 200000, 500000];
cs = cell(9,1);
for i=1:7
    cs(i) = {sprintf('%d', handles.TabNS(i))};
end
set(handles.lbNSamples, 'String', cs, 'Value', 5);

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes TestAcqScope wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Plota as amostras aquisitadas
function plot_samples(handles)
if handles.n > 0
    figure (handles.hPlot);
    plot (handles.B(1, 1:handles.n)');
    hold on
    plot (handles.B(2,1:handles.n)', 'red');
    %xlim ([0 handles.n]);
    grid on
end



% --- Tratamento do evento do timer.
function update_samples(obj, event, hObject)
handles = guidata(hObject);
if handles.fAcqOn
    if ~handles.fTimer
        handles.fTimer = true;
        guidata(hObject, handles);
        [r, ieStatus, nSamples, iLast, nSampGot, handles.BufN, handles.BufDi] =
handles.h.GetSamplesEx(2,handles.BufN, handles.BufDi);
        % Update handles structure
        if (~ r) || (ieStatus ~= 0)
            btStopAcq_Callback(hObject, event, handles);
            handles = guidata(hObject);
        else
            if (handles.n + nSampGot) >= handles.nSampProg
                btStopAcq_Callback(hObject, event, handles);
                handles = guidata(hObject);
                nSampGot = handles.nSampProg - handles.n;
            end
            % copia as amostras
            for i = 1:nSampGot
                handles.B(:, handles.n+i) = handles.BufN(:,i);
            end
            handles.n = handles.n + nSampGot;
            sProgress = sprintf ('%d / %d', handles.n, handles.nSampProg);
            set (handles.lblProgress, 'String', sProgress);
            if handles.n >= handles.nSampProg
                plot_samples(handles);
            else
                % Não altere o plot para incluir mais pontos ou gráficos.
                % O Matlab pode não dar conta disso em tempo real.
                figure (handles.hPlot);
                hold off
                if handles.n <= 500
                    plot (handles.B(1,1:handles.n)');
                else
```

```
                    plot (handles.B(1,handles.n-500:handles.n)');
                end
                YA = ylim;
                if handles.YS(1) > YA(1)
                    handles.YS(1) = YA(1);
                end
                if handles.YS(2) < YA(2)
                    handles.YS(2) = YA(2);
                end
                xlim ([1 500]);
                ylim (handles.YS);
                grid on
            end

        end
        handles.fTimer = false;
    end
    guidata(hObject, handles);
end


% --- Outputs from this function are returned to the command line.
function varargout = TestAcqScope_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in btCreateCOM.
function btCreateCOM_Callback(hObject, eventdata, handles)
% hObject    handle to btCreateCOM (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.h = actxserver('LynxADS2500.LynxDriver');
set (handles.lblWarning, 'Visible', 'On');
handles.fAcqOn = false;
guidata(hObject, handles);
sPcIP  = get(handles.edPcIP, 'String');
sDevIP = get(handles.edDevIP, 'String');
handles.fOpened = handles.h.Connect (sPcIP, sDevIP);
set (handles.lblWarning, 'Visible', 'Off');
if handles.fOpened
    set (handles.btCreateCOM, 'Enable', 'Off');
    set (handles.btReleaseCOM, 'Enable', 'On');
    set (handles.btStartAcq, 'Enable', 'On');
    start(handles.tmr);
else
    handles.h.delete;
end
guidata(hObject, handles);


% --- Executes on button press in btReleaseCOM.
function btReleaseCOM_Callback(hObject, eventdata, handles)
% hObject    handle to btReleaseCOM (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

stop(handles.tmr);
if handles.fAcqOn
    handles.fAcqOn = false;
    handles.h.AcqSetup(0, 100);
end
if handles.fOpened
    handles.h.Disconnect;
    handles.h.delete;
    handles.fOpened = false;
```

```
end
set (handles.btCreateCOM, 'Enable', 'On');
set (handles.btReleaseCOM, 'Enable', 'Off');
set (handles.btStartAcq, 'Enable', 'Off');
set (handles.btStopAcq, 'Enable', 'Off');
guidata(hObject, handles);



% --- Executes on button press in btStartAcq.
function btStartAcq_Callback(hObject, eventdata, handles)
% hObject    handle to btStartAcq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

if handles.fOpened
    handles.h.AiRangeIndex = 0;   % Select +/- 10V

    handles.h.ClearICM;
    handles.h.InsertICM (0, 1);
    handles.h.InsertICM (0, 2);
    handles.h.InsertICM (0, 3);
    handles.h.InsertICM (0, 5);
    handles.h.InsertICM (1, 1);
    handles.Fs = handles.h.AcqSetup (3, 8000);

    handles.nSignals  = handles.h.nAiSignals + handles.h.nCtrSignals;
    figure (handles.hPlot);
    handles.YS = [1 -1];
    % cria o buffer para o GetSamples
    clear handles.BufN
    clear handles.BufDi
    handles.BufN = zeros(handles.nSignals, 65536, 'double');
    handles.BufDi = zeros(handles.h.nDiSignals, 65536, 'uint32');

    iNS = get(handles.lbNSamples,'Value');
    handles.nSampProg = handles.TabNS(iNS);
    handles.n         = 0;
    % cria o buffer para armazenar as amostras: nSampProg samples x nSignals
    clear handles.B;
    handles.B = zeros(handles.nSignals, handles.nSampProg, 'double');

    if handles.h.StartCapture (handles.nSampProg)
        set (handles.lbNSamples, 'Enable', 'Off');
        set (handles.btStartAcq, 'Enable', 'Off');
        set (handles.btStopAcq, 'Enable', 'On');
        handles.fAcqOn = true;
        guidata(hObject, handles);
        %start(handles.tmr);
    end
end
guidata(hObject, handles);


% --- Executes on button press in btStopAcq.
function btStopAcq_Callback(hObject, eventdata, handles)
% hObject    handle to btStopAcq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%stop(handles.tmr);
guidata(hObject, handles);

handles.h.StopCapture;
handles.h.AcqSetup(0, 100);
handles.fAcqOn = false;
set (handles.lbNSamples, 'Enable', 'On');
set (handles.btStartAcq, 'Enable', 'On');
set (handles.btStopAcq, 'Enable', 'Off');
% atualiza o gráfico com todas as amostras
plot_samples(handles);
guidata(hObject, handles);
```

```
% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

stop(handles.tmr);
if handles.fAcqOn
    handles.h.AcqSetup(0, 100);
    handles.fAcqOn = false;
end
if handles.fOpened
    handles.h.Disconnect;
    handles.h.delete;
    handles.fOpened = false;
end
delete(handles.tmr);
if ishandle(handles.hPlot)
    close (handles.hPlot);
end
% Hint: delete(hObject) closes the figure
delete(hObject);

% --- Executes on selection change in lbNSamples.
function lbNSamples_Callback(hObject, eventdata, handles)
% hObject    handle to lbNSamples (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns lbNSamples contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from lbNSamples


% --- Executes during object creation, after setting all properties.
function lbNSamples_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lbNSamples (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edDevIP_Callback(hObject, eventdata, handles)
% hObject    handle to edDevIP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edDevIP as text
%        str2double(get(hObject,'String')) returns contents of edDevIP as a double


% --- Executes during object creation, after setting all properties.
function edDevIP_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edDevIP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function edPcIP_Callback(hObject, eventdata, handles)
% hObject    handle to edPcIP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edPcIP as text
%        str2double(get(hObject,'String')) returns contents of edPcIP as a double


% --- Executes during object creation, after setting all properties.
function edPcIP_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edPcIP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

6.1. Requisitos

O acesso ao automation server *LynxADS2500.LynxDriver* em Pytyon requer a versão do Python para Windows na plataforma Intel.

São necessários também os seguintes packages do Python:

- **pywin32**
  O package *pywin32* é um pacote com extensões Python para Microsoft Windows que disponibiliza acesso ao Win32 API e o suporte a objetos COM. A versão do *pywin32* utilizada no desenvolvimento dos exemplos dessa documentação é o `pywin32-223-cp37-cp37m-win32.whl`.
- **numpy**
  O package *numpy* é um pacote desenvolvido para processamento eficiente de arrays multi dimensional. A versão do *numpy* utilizada no desenvolvimento dos exemplos dessa documentação é o `numpy-1.15.0-cp37-none-win32.whl`.

Esses pacotes podem ser obtidos no site do PyPi.org (Python Package Index):

https://pypi.org/

O PyPi é um repositório de software para a linguagem de programação Python.

6.2. Instalação dos Pacotes win32py e numpy

Esse tópico pode ser pulado se você já tiver os *packages win32py* e *numpy* instalados no seu computador.

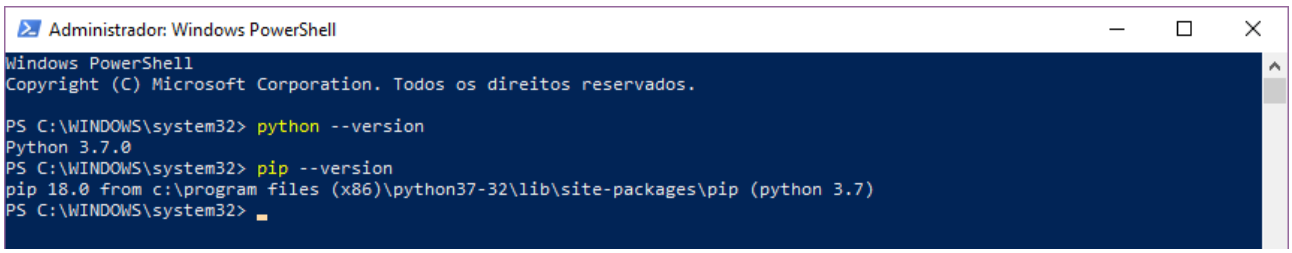
Para a instalação desses pacotes podem ser seguidas as instruções da documentação do Python.

Se preferir, pode-se instalar os pacotes ***win32py*** e ***numpy*** na janela do *PowerShell(Admin)* do Windows 10 ou na janela do *Prompt de Comando* do Windows 7 ou 8. Em ambos os casos deve-se executar esses comandos com privilégio de administrador.

Antes executar os comandos para instalação dos pacotes, deve-se verificar se o *Python* e o *pip* estão instalados no seu computador.

Para verificar se o *Python* e o *pip* se encontram instalados, execute os seguintes comandos no *PowerShell(Admin)* do Windows.

```
python --version
pip --version
```



Se o *PowerShell* indicar falha na execução dos comandos citados, pode ser que o *Python* não foi instalado ou o diretório do *Python* e/ou *pip* não estão no *path* do Windows.

Na configuração padrão do instalador do Python a opção de inclusão do diretório do Python no *path* do Windows não é habilitada.

Se a execução dos comandos foi bem sucedida, execute os comandos abaixo no *PowerShell(Admin)* do Windows.

```
pip install 'c:\user\Lauro\Downloads\Python\pywin32-223-cp37-cp37m-win32.whl'

pip install 'c:\user\Lauro\Downloads\Python\numpy-1.15.0-cp37-none-win32.whl'
```

O diretório e o nome dos arquivos citados no comando `pip install` são apenas exemplos. Substitua-os pelo diretório e nomes dos arquivos dos pacotes *win32py* e *numpy* que você baixou do site da *pypi.org*.

6.3. Iniciação do Python

Para acessar o pacote win32py e numpy no ambiente do Python, é necessário importar as referências a esses pacotes. Para isso execute os seguintes comandos na janela de prompt de comando do Python:

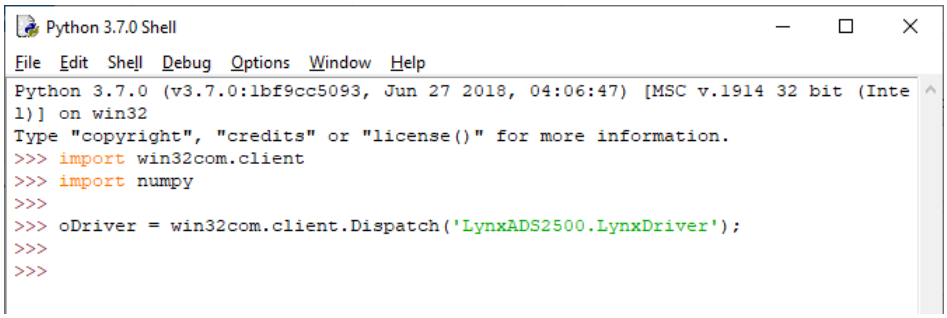
```
import win32com.client
import numpy
```

6.4. Criação da Instância do LynxADS2500.LynxDriver

A instância do *do driver do ADS2500* é criada através do seguinte comando em Python.

```
oDriver = win32com.client.Dispatch('LynxADS2500.LynxDriver');
```

A figura abaixo ilustra a janela de comandos do Python após importação das referências aos pacotes *win32py* e *numpy* e a criação da instância do drier do ADS2500.



Após criar a instância do seiver *LynxADS2500.LynxDriver*, tem-se acesso às propriedades e aos métodos da interface *ILynxDriver* implementada pelo automation server.

A variável ***oDriver*** é o handle da instância do automation server criado pela chamada da *win32com.client.Dispatch*.

No decorrer desta documentação será utilizada a variável ***oDriver*** como a instância do driver do ADS2500.

6.5. Acesso às Propriedades da Interface ILynxDriver

Para acessar uma propriedade da interface *ILynxDriver*, basta referenciar o nome da propriedade através do *handle* da instância do automation server *LynxADS2500.LynxDriver*.

A tabela seguinte lista as propriedades da interface *ILynxDriver*.

Propriedade	Type	R/W	Index	Descrição
LastErrorCode	int16	R		Código do último erro
IsConnected	boolean	R		Indica se está conectado com os equipamentos
sPcIP	string	R		Endereço IP da interface de rede do computador
sHostIP	string	R		Endereço IP do equipamento
DeviceModel	string	R		Modelo do hardware de aquisição de dados
DeviceSN	string	R		Número de série do hardware
DeviceTag	string	R		Tag do hardware
EnumStart1 [1]	boolean	R		Indica se a numeração dos canais se inicia em 1.
nAiChannels	int16	R		Número de canais A/D
nAiRange	int16	R		Número de opções de faixa de entrada do conversor A/D
AiRange	double	R	Index	Valor da Index-ésima faixa de entrada do conversor A/D. Index de 0 a nAiRange-1. Valor positivo indica faixa unipolar. Por exemplo, o valor 10 indica uma faixa de entrada de 0 a 10. Valor negativo indica faixa bipolar. Por exemplo, o valor -10 indica uma faixa de entrada de -10 a 10.
AiRangeUnit	String	R	Index	Unidade da Index-ésima faixa do conversor A/D Index de 0 a nAiRange-1
AiRangeIndex	uint16	R/W		Seleciona a faixa de entrada do conversor A/D
nAoChannels	Int16	R		Número de canais D/A
nAoRange	Int16	R		Número de opções de faixa de faixa de saída do conversor D/A
AoRange	double	R	Index	Valor da Index-ésima faixa de saída do conversor D/A. Index de 0 a nAoRange-1 Valor positivo indica faixa unipolar. Por exemplo, o valor 10 indica uma faixa de saída de 0 a 10. Valor negativo indica faixa bipolar. Por exemplo, o valor -10 indica uma faixa de saída de -10 a 10.
AoRangeUnit	string	R	Index	Unidade da Index-ésima faixa de saída do conversor D/A Index de 0 a nAoRange-1
AoRangeIndex	uint16	R/W		Seleciona a faixa de saída do conversor D/A
nCtrChannels	int16	R		Número de canais de contagem
nCtrBits	int16	R		Número de bits dos canais de contagem
nFResol	int16	R		Número de opções de resolução na medição de frequência nos canais de contagem de pulso.
FResol	double	R	Index	Valor da Index-ésima resolução na medição de frequência nos canais de contagem de pulso (em Hz) Index de 0 a Fresol-1
FResolIndex	unsigned int16	R/W		Seleciona a resolução da medição de frequência nos canais de contagem de pulso
nTResol	int16	R		Número de opções de resolução na medição de período nos canais de contagem de pulso.
TResol	double	R	Index	Valor da Index-ésima resolução da medição de período nos canais de contagem de pulso (em microsegundos) Index de 0 a TResol-1

Propriedade	Type	R/W	Index	Descrição
TResolIndex	unt16	R/W		Seleciona a resolução da medição de período nos canais de contagem de pulso
nDiPorts	int16	R		Número de ports de entrada digital
nDiBits	int16	R		Número de bits por port entrada digital
nDoPorts	int16	R		Número de ports de saída digital
nDoBits	int16	R		Número de bits por port de saída digital
IsAcqOn	boolean	R		Indica se a aquisição de sinais está ativa
SampleFreq	double	R		Frequência de amostragem
nAiSignals	int16	R		Número de canais A/D habilitados para aquisição
nCtrSignals	int16	R		Número de canais de contagem habilitados para aquisição
nDiSignals	int16	R		Número de ports DI habilitados para aquisição
tSample	int64	R		Número de amostras aquisitadas desde o início da aquisição após a chamada do método *AcqSetup* com comando de partida da aquisição. Equivale ao contador de intervalos de amostragem e pode ser útil para determinar quando há um dado novo nas aplicações que utilizam o modo de aquisição *Single Burst*.
nSampProg	Int32	R		Número de amostras a serem aquisitadas, programada no método *StartCapture*
nSamples	Int32	R		Número de amostras aquisitadas por canal do total de *nSampProg*
AiName [1, 2, 3]	string	R/W	Channel	Nome do sinal canal de entrada analógica Channel de 1 a nAiChannels
AiUnit [1, 2, 3]	string	R/W	Channel	Unidade de engenharia do sinal no canal de entrada analógica Channel de 1 a nAiChannels
AiHiLim [1, 2, 3]	double	R/W	Channel	Limite superior do sinal no canal de entrada analógica em unidade de engenharia Channel de 1 a nAiChannels
AiLoLim [1, 2, 3]	double	R/W	Channel	Limite inferior do sinal no canal de entrada analógica em unidade de engenharia Channel de 1 a nAiChannels
CtrName [1, 2, 3]	string	R/W	Channel	Nome do sinal no canal de entrada de pulso Channel 1 a nCtrChannels
CtrUnit [1, 2, 3]	string	R/W	Channel	Unidade de engenharia do sinal no canal de entrada de pulso Channel 1 a nCtrChannels
CtrFactor [1, 2, 3]	double	R/W	Channel	Fator de conversão do sinal no canal de contagem de pulso para unidade de engenharia. Valor em unidade de engenharia por pulso. Channel 1 a nCtrChannels
DiBitName [1, 2, 3]	string	R/W	Channel, Bit	Nome do bit de entrada digital Channel de 1 a nDiPorts
DiBitLow [1, 2, 3]	stirng	R/W	Channel, Bit	Valor para nível low de bit de entrada digital Channel de 1 a nDiPorts
DiBitHigh [1, 2, 3]	string	R/W	Channel, Bit	Valor para nível high de bit de entrada digital Channel de 1 a nDiPorts
AoName [1, 2, 3]	string	R/W	Channel	Nome do sinal no canal de saída analógica Channel de 1 a nAoChannels
AoUnit [1, 2, 3]	string	R/W	Channel	Unidade de engenharia do sinal no canal de saída analógica Channel de 1 a nAohannels
AoHiLim [1, 2, 3]	double	R/W	Channel	Limite superior do canal de saída analógica em unidade de engenharia Channel de 1 a nAoChannels
AoLoLim [1, 2, 3]	double	R/W	Channel	Limite inferior do canal de saída analógica em unidade de engenharia Channel de 1 a nAoChannels
DoBitName [1, 2, 3]	string	R/W	Channel, Bit	Nome do bit de saída digital Channel de 1 a nDoPorts

Propriedade	Type	R/W	Index	Descrição
DoBitLow [1, 2, 3]	string	R/W	Channel, Bit	Valor para nível low de bit de saída digital Channel de 1 a nDoPorts
DoBitHigh [1, 2, 3]	string	R/W	Channel, Bit	Valor para nível high de bit de saída digital Channel de 1 a nDoPorts

Notas:

1. Esses métodos foram incluídos na versão 2.0 da interface *ILynxDriver*.
2. Os valores dessas propriedades são carregados do ADS2500 durante a chamada do método *Connect*. É mais fácil utilizar o assistente do *ADS2500* através do *Lynx@Net* para configurar os canais. Após a configuração ter sido salva no ADS2500, ela é carregada pelo driver através do método *Connect*.
3. A leitura dessas propriedades com índice pode executada de duas maneiras. Segue abaixo o exemplo do acesso da propriedade *AiName*:

```
r = oDriver.AiName(1)
r = oDriver.GetAiName(1)
```

Para atribuir o valor 'Force' na propriedade *AiName* do canal 1, utilize o seguinte comando:

```
r = oDriver.SetAiName(1, 'Force')
```

A figura abaixo ilustra um exemplo de consultas a propriedades da interface do automation server.

6.6. Acesso aos Métodos da Interface ILynxDriver

Os métodos da interface com automation server *LynxADS2500.LynxDriver* são acessados através do handle da sua instância.

Por exemplo, na linha abaixo é chamado o método *Connect*.

```
r = oDriver.Connect('', '192.168.1.125');
```

A tabela seguinte lista os métodos da interface *ILynxDriver*.

Função	Descrição
Connect	Estabelece a conexão o ADS2500.
Disconnect	Fecha a conexão com o ADS2500.
QueryDeviceID	Obtém identificação do equipamento
ProgCtr	Programa modo de operação de canal de contagem de pulso
ClearICM	Limpa a memória de canais
InsertICM	Insere canal na memória de canais
AcqSetup	Configura a aquisição de sinais
StartCapture	Inicia a captura da aquisição de sinais
StopCapture	Finaliza a captura da aquisição de sinais
GetSignalMap	Mapeamento dos sinais aquisitados com os canais físicos
GetSamples	Leitura das últimas amostras aquisitadas de cada sinal
GetSamplesEx [1]	Leitura das últimas amostras aquisitadas de cada sinal
GetLast	Leitura da última amostra aquisitada de cada sinal
GetLastEx [1]	Leitura da última amostra aquisitada de cada sinal
ReadAi	Leitura de canal de entrada analógica
ReadAiEx [1]	Leitura de canal de entrada analógica
ReadCtr	Leitura de canal de contagem de pulso
ReadCtrEx [1]	Leitura de canal de contagem de pulso
ReadDi	Leitura de port de entrada digital
ReadBitDi	Leitura de bit de port de entrada digital
ReadBitDiEx [1]	Leitura de bit de port de entrada digital
ReadDo	Leitura de port de saída digital (última escrita)
ReadBitDo	Leitura de bit de port de saída digital
ReadBitDoEx [1]	Leitura de bit de port de saída digital
WriteDo	Escrita em port de saída digital
WriteAo	Escrita em canal de saída analógica
WriteAoEx [1]	Escrita em canal de saída analógica
GetSnProp [1]	Informa o nome e a unidade de engenharia de um sinal habilitado para aquisição de sinais (canal A/D ou de contagem de pulso)
SetAiSetup [1]	Especifica o nome, a unidade de engenharia e os limites superior e inferior de um canal de entrada analógica
GetAiSetup [1]	Informa o nome, a unidade de engenharia e os limites superior e inferior de um canal de entrada analógica
SetAoSetup [1]	Especifica o nome, a unidade de engenharia e os limites superior e inferior de um canal de saída analógica
GetAoSetup [1]	Informa o nome, a unidade de engenharia e os limites superior e inferior de um canal de saída analógica
SetCtrSetup [1]	Especifica o nome, a unidade de engenharia e o fator para conversão para unidade de engenharia de um canal de entrada de pulso

Função	Descrição
GetCtrSetup [1]	Informa o nome, a unidade de engenharia e o fator para conversão para unidade de engenharia de um canal de entrada de pulso

Notas:

1. Esses métodos foram incluídos na versão 2.0 da interface *ILynxDriver*.
2. Os valores das propriedades associadas a esses métodos são carregados do ADS2500 durante a chamada do método *Connect*. É mais fácil utilizar o assistente do *ADS2500* através do *Lynx@Net* para configurar os canais. Após a configuração ter sido salva no ADS2500, ela é carregada pelo driver através do método *Connect*.
3. A alteração dos valores das propriedades associadas a esses métodos só tem efeito durante a operação on line.

Neste tópico serão descritos em *Python* alguns métodos da interface *ILynxDriver*. A descrição completa dos métodos é apresentada em Delphi no tópico *3.2.Métodos da Interface ILynxDriver*.

6.6.1. Método Connect

Após criação de uma a instância do automation *LynxADS2500.LynxDriver*, deve-se estabelecer a conexão com o ADS2500. A conexão é realizada através da chamada do método *Connect* da interface *ILynxDriver*.

```
r = oDriver.Connect(sPcIP, sHostIP);
```

Parâmetro	Descrição
sPcIP	Parâmetro de entrada do tipo string. Especifique neste parâmetro o endereço IP da interface de rede do computador que será utilizada para conexão com o ADS2500. Este parâmetro pode ser deixado em branco. Nesse caso, o driver tentará se conectar com o ADS2500 através de uma das placas de rede do computador.
sHostIP	Parâmetro de entrada do tipo string. Especifique neste parâmetro o endereço IP do ADS2500. Este parâmetro pode ser deixado em branco. O endereço padrão do ADS2500 é `192.168.1.125`.
r	Valor retornado do tipo boolean A função retorna *true* se a conexão com o ADS2500 foi estabelecida com sucesso ou *false* se a conexão não pode ser efetivada. Umas das causas da falha de conexão são: endereço IP errado, equipamento desligado, rota não estabelecida se o equipamento está em outra subrede, bloqueio do firewall e excedeu o número de conexões simultâneas do ADS2500.

6.6.2. Método ReadAiEx

Este método é utilizado para a leitura de um canal A/D.

```
r = oDriver.ReadAiEx(Channel, SampleFormat);
```

Este método retorna o valor da última amostra lida do canal A/D especificado. O valor da amostra é normalizado em ±1.0 (consulte o tópico 2 deste guia do usuário).

Parâmetro	Descrição
Channel	Parâmetro de entrada do tipo ***int16***. Número do canal A/D do equipamento. No ADS2500 a numeração dos canais é a partir de 1. Se o canal A/D não estiver habilitado ou a aquisição de sinais não estiver ativa, o método valor zero.
SampleFormat	Parâmetro de entrada do tipo ***int16***. Especifica o formato com o qual a amostra da entrada analógica será retornada: • 0: normalizado (amostras normalizadas em +/- 1.0) • 1: unidade de entrada (volts) • 2: unidade de engenharia
r	Valor retornado em ***double***. O valor retornado pelo *ReadSample* é o valor da amostra do canal especificado.

6.6.3. Método GetLastEx

Este método retorna o valor da última amostra aquisitada de cada sinal de entrada. As amostras das entradas analógicas e de contagem de pulso são retornadas no formato especificado no parâmetro *sf*.

```
r, st, b, d = oDriver.GetLastEx(st, sf, b, d);
```

Parâmetro	Descrição
st	Parâmetro de saída do tipo ***uint16***. Status da execução do método *GetLastEx*. Veja no enumerador *ieStatus* os valores possíveis retornados pelo método.
sf	Parâmetro de entrada do tipo ***int16***. Especifica o formato com o qual a amostra da entrada analógica será retornada: • 0: normalizado (amostras normalizadas em +/- 1.0) • 1: unidade de entrada (volts) • 2: unidade de engenharia
b	Parâmetro de entrada e saída do tipo ***Safearray***. No parâmetro ***b*** deve ser passado um array do tipo ***double*** com tamanho maior ou igual *igual a nAiSignals + nCtrSignals.*.Ou seja, maior que o número de canais A/D e de contagem de pulso habilitados para aquisição..
d	Parâmetro de entrada e saída do tipo ***Safearray***. No parâmetro ***d*** deve ser passado um array do tipo ***unsigned long*** com tamanho maior ou igual a *nDiSignals*. Ou seja, maior ou igual ao número de ports DI habilitados para aquisição..
r	A função retorna *true* se a leitura foi realizada com sucesso. Consulte o valor de *st* se o valor retornado for *false*.

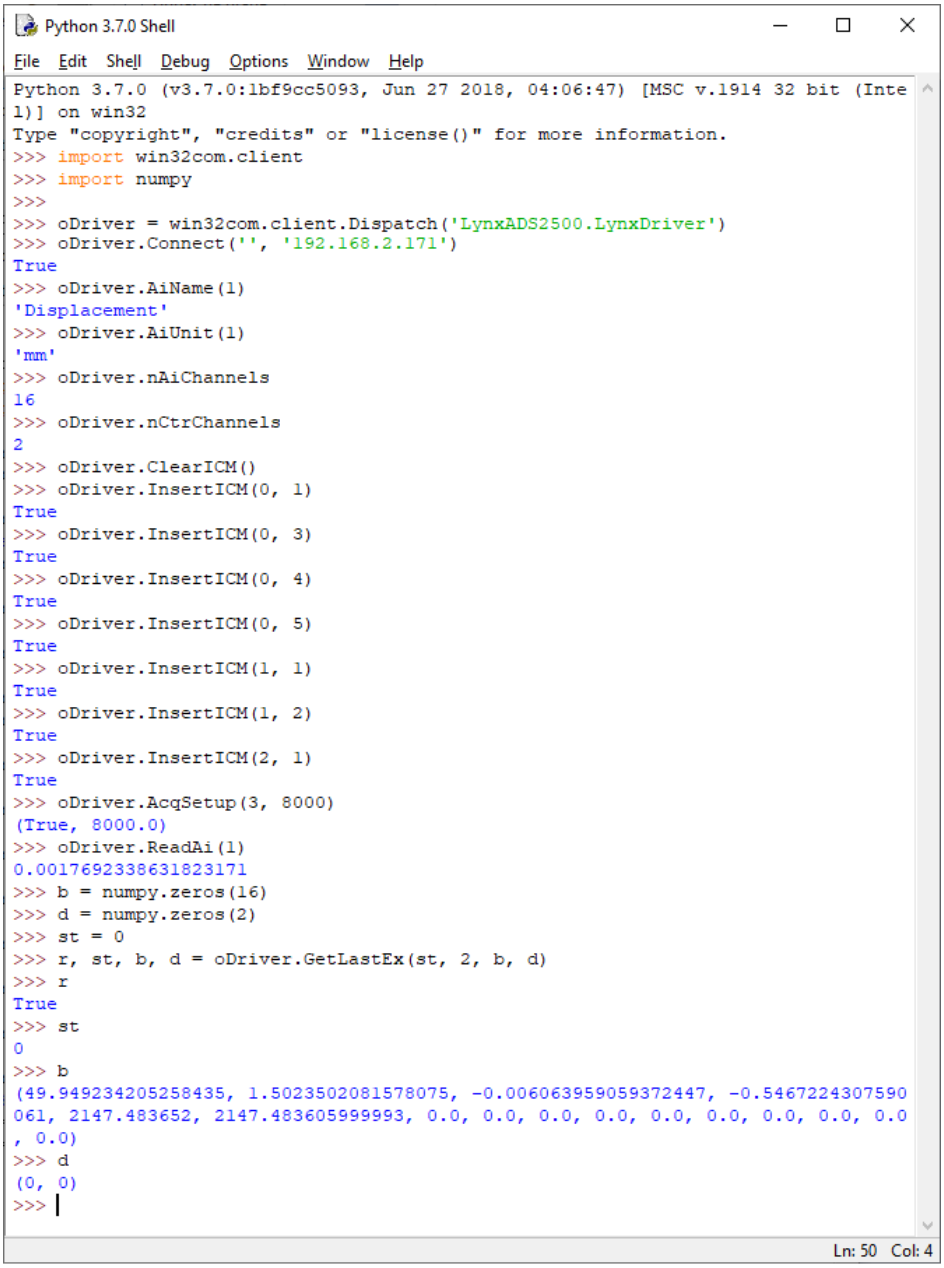
O vetor passado no parâmetro *b* do método GetLastEx deve ser do tipo *double* e ter tamanho suficiente para as leituras realizadas através do *GetLastEx*. O parâmetro *b* é de entrada e saída. No entanto, o suporte ao *Activex* no Python utiliza apenas as informações do tipo e da dimensão do vetor para efetuar a conversão para o tipo safearray. Na saída do método *GetLastEx* o Python não transfere os valores retornados no saferray para o parâmetro *b*. O conteúdo do saferray é

transferido pelo Python para um vetor diferente de saída.

Por essa característica do Python, o vetor *b* pode ser criado uma única vez e com tamanho adequado para a aplicação. O comando abaixo exemplifica a criação do vetor *b* do tipo *double* e com tamanho 16. O vetor é criado através de uma função do package ***numpy***.

```
b = numpy.zeros(16);
d = numpy.zeros(2, 'uint32');
```

A figura seguinte ilustra um exemplo de chamada do método *GetLastEx*. Nesse exemplo foram programados para aquisição de sinais os canais A/D 1, 3, 4 e 5, os canais de contagem de pulso CTR 1 e 2 e o port de entradas digitais. A frequência de amostragem foi programada em 8000 Hz.



6.6.4. Método GetSamplesEx

Este método retorna um conjunto de amostras aquisitadas desde a última chamada do método ou que ainda não foram lidas. O número de amostras retornadas por canal é limitado pelo tamanho dos safearrays.

```
r, st, ns, iL, ng, B, D = oFileTS.GetSamplesEx(st, sf, ns, iL, ng, B, D);
```

Este método é utilizado em conjunto com o método *StartCapture* que deve ser executado antes da primeira chamada do *GetSamplesEx*. Na chamada do *StartCapture* é passado o número de amostras a ser aquisitado por canal. As amostras aquisitadas são armazenadas num buffer circular e devem ser lidas periodicamente através do *GetSamplesEx*.

Parâmetro	Descrição
st	Parâmetro de saída do tipo ***uint16***. Status da execução do método *GetLastEx*. Veja no enumerador *ieStatus* os valores possíveis retornados pelo método.
sf	Parâmetro de entrada do tipo ***int16***. Especifica o formato com o qual a amostra da entrada analógica será retornada: • 0: normalizado (amostras normalizadas em +/- 1.0) • 1: unidade de entrada (volts) • 2: unidade de engenharia
ns	Parâmetro de saída do tipo ***int32***. Número de amostras aquisitadas por canal desde o início da aquisição de sinais.
iL	Parâmetro de saída do tipo ***int32***. Índice da última amostra retornada por esta chamada do método.
ng	Parâmetro de saída do tipo ***int32***. Número de amostras por canal transferidas para os buffers.

B	Parâmetro de entrada e saída do tipo ***Safearray***.
	O saferray deve ter as seguintes características:
	• Tipo de elemento: Double
	• Número de dimensões: 1 ou 2.
	• Vetor de N elementos.
	  ○ N é o número total de amostras que o vetor ou a matriz comporta.
	  ○ O número máximo de amostras por canal que o safearray aceita é igual a N div (*nAiSignals* + *nCtrSignals*).

D	Parâmetro de entrada e saída do tipo ***Safearray***.
	O saferray deve ter as seguintes características:
	• Tipo de elemento: uint32.
	• Número de dimensões: 1 ou 2.
	• Vetor de N elementos.
	  ○ N é o número total de amostras que o vetor ou a matriz comporta.
	  ○ O número máximo de amostras por canal que o safearray aceita é igual a N div *nDiSignals*.

r	A função retorna true se a leitura foi realizada com sucesso. Consulte o valor de st se o valor retornado for false.

Os parâmetros *B* e *D* poderiam ser definidos como matrizes. No entanto, o suporte a COM do Python só converte safearrays de uma dimensão.

As amostras das entradas analógicas e de contagem de pulso são retornadas no formato

especificado no parâmetro *SampleFormat*.

O vetor passado no parâmetro *B* do método *GetSamplesEx* deve ser do tipo *double* e ter tamanho suficiente para as leituras realizadas através do *GetSamplesEx*. O parâmetro *B* é de entrada e saída. No entanto, o suporte ao *Activex* no Python utiliza apenas as informações do tipo e da dimensão do vetor para efetuar a conversão para o tipo safearray. Na saída do método *GetSamplesEx* o Python não transfere os valores retornados no saferray para o parâmetro *B*. O conteúdo do saferray é transferido pelo Python para um vetor diferente de saída.

Por essa característica do Python, os vetores *B* e *D* podem ser criados uma única vez e com tamanho adequado para a aplicação. No exemplo abaixo os vetores B e D são criados, respectivamente, com tamanho 200 e 50.

```
B = numpy.zeros(200);
D = numpy.zeros(50, 'uit32');
```

A figura seguinte ilustra um exemplo de chamada do método *GetSamplesEx*. Nesse exemplo foram programados para aquisição de sinais os canais A/D 1, 3, 4 e 5, os canais de contagem de pulso CTR 1 e 2 e o port de entradas digitais. A frequência de amostragem foi programada em 2000 Hz e a captura foi programada para 10000 amostras por canal.