

PUC de Propósito Especial

Versão 1.0
06 de agosto de 2014

Bruno Martins
bruno.martins@lnls.br
Grupo de Controle

Histórico de Revisões

Revisão	Mudanças
1.0 06/08/2014	<ul style="list-style-type: none">• Versão Inicial.

Sumário

1 Visão Geral do Hardware.....	1
2 Teoria de Operação.....	3
2.1 Variáveis analógicas.....	3
2.2 Variáveis digitais.....	3
2.3 Procedimento Síncrono.....	3
3 Entidades BSMP.....	5
3.1 Variável: Placas Detectadas.....	5
3.2 Variável: Estado do Procedimento Síncrono.....	5
3.3 Variável: Configuração do Procedimento Síncrono.....	6
3.4 Variáveis: Entradas Analógicas.....	6
3.5 Variáveis: Saídas Analógicas.....	6
3.6 Variáveis: Entradas Digitais.....	6
3.7 Variáveis: Saídas Digitais.....	7
3.8 Curva: RAM.....	7
3.9 Curva: Flash.....	7
3.10 Função: Reset.....	7
3.11 Função: Iniciar Procedimento Síncrono.....	7
3.12 Função: Parar Procedimento Síncrono.....	7
3.13 Função: Pausar Procedimento Síncrono.....	8
3.14 Função: Passo Procedimento Síncrono.....	8
4 Biblioteca pypucsp.....	9
4.1 Instanciação.....	9
4.2 Placas detectadas.....	9
4.3 Entradas e saídas analógicas.....	9
4.3.1 Obtendo uma referência.....	10
4.3.2 Leitura.....	10
4.3.3 Escrita.....	10
4.4 Entradas e saídas digitais.....	10
4.4.1 Obtendo uma referência.....	10
4.4.2 Leitura.....	10
4.4.3 Escrita.....	11
4.4.4 Operações binárias.....	11
4.5 Reset.....	11
4.6 Procedimento Síncrono.....	11
4.7 Procedimento Síncrono: Estado.....	11
4.8 Procedimento Síncrono: Curvas.....	12
4.8.1 Leitura.....	12
4.8.2 Escrita.....	12
4.9 Procedimento Síncrono: Configuração.....	12
4.9.1 Classe SyncConfig.....	12
4.9.2 Ler.....	13
4.9.3 Escrever.....	13
4.10 Procedimento Síncrono: Execução.....	13
5 Interface qtClient.....	14
5.1 Conexão e Status.....	15

5.2 Escrita e leitura em variáveis.....	15
5.3 Configuração do procedimento síncrono.....	15
5.4 Geração de curvas.....	16
5.5 Execução do procedimento síncrono.....	17
5.6 Leitura de curvas.....	18

1 Visão Geral do Hardware

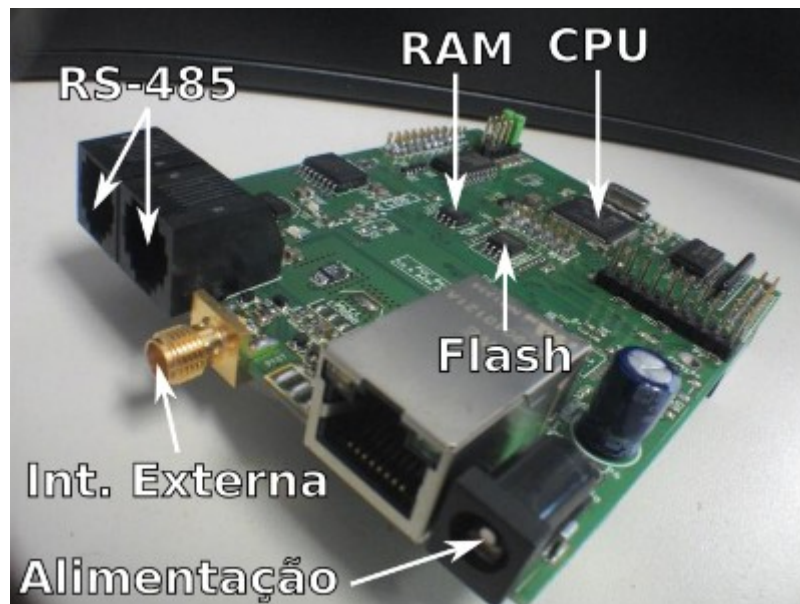


Figura 1- Componentes e conectores da base da PUC

A PUC é um *hardware* modular, composto de uma base e de placas de extensão. A base (versão v1.4 ARM) é composta de um microcontrolador, uma memória RAM volátil e uma memória Flash não-volátil. Além disso a base possui conectores para alimentação (9V), comunicação serial RS-485 (RJ-22), comunicação Ethernet (não utilizada nesse *firmware*) e entrada de interrupção (SMA). A base e seus componentes principais podem ser vistos na Figura 1.

A comunicação com a PUC é feita sobre RS-485. O protocolo utilizado é o protocolo BSMP. O *baud-rate* é de 6 Mbps. O endereço serial da PUC é selecionado com *jumper*s nos pinos mostrados na Figura 2. Nela, a PUC está configurada com o endereço 2. Endereços válidos são de 1 a 31.



Figura 2- Pinos de endereço da base da PUC

Placas de extensão podem ser empilhadas na base, aumentando suas funcionalidades. No momento da escrita deste documento, há dois tipos de placas de extensão: analógica e digital. A analógica contém um conversor A/D e um conversor D/A, ambos com 18 bits de precisão. A placa digital contém uma entrada e uma saída digitais, ambas com 8 bits. As placas de extensão também têm seu endereço selecionado por *jumper*s, como mostra a Figura 3 (na qual o endereço selecionado é 3). Os endereços válidos são de 0 a 3.



Figura 3- Pinos de endereço de uma placa de extensão

2 Teoria de Operação

A PUC de Propósito Especial tem duas funções básicas:

1. Escrever e ler entradas e saídas analógicas e digitais
2. Procedimento síncrono: conversão de valores em memória para uma saída analógica e de uma entrada analógica para memória de forma sincronizada.

2.1 Variáveis analógicas

As variáveis analógicas aparecem quando uma placa analógica está conectada à PUC. Cada placa analógica adiciona uma entrada e uma saída. Seus valores variam de -10 V a 10 V, e os conversores têm precisão de 18 *bits*.

2.2 Variáveis digitais

Assim como as analógicas, as variáveis digitais aparecem quando uma placa digital está conectada à PUC. Cada placa digital adiciona uma entrada e uma saída à PUC, de 8 *bits* cada.

2.3 Procedimento Síncrono

A PUC é provida de duas memórias para armazenar pontos de curvas: Flash e RAM. A Flash é uma memória não volátil, utilizada para armazenar pontos a serem enviados para a saída analógica. Já a RAM é utilizada para armazenar pontos lidos da entrada analógica. Apesar de a Flash ter 8MB de capacidade de armazenamento, apenas 128 kB são disponibilizados pela versão atual do *firmware*, para se equivaler à capacidade da RAM.

A idéia do procedimento síncrono é que, de maneira sincronizada a partir de um *clock* interno ou externo, um ponto lido pela entrada analógica seja armazenado na RAM e um ponto na memória Flash seja escrito na saída analógica, sequencialmente.

A precisão dos pontos pode ser selecionada entre 16 *bits* (pontos de 2 *bytes* nas memórias) e 18 *bits* (pontos de 4 *bytes* nas memórias). Assim, cada memória pode armazenar 65536 pontos de 16 *bits* ou 32768 pontos de 18 *bits*.

A fonte de *clock* pode ser selecionada entre Interna, na qual um *timer* do microcontrolador é utilizado, Externa, na qual uma fonte de *clock* deve ser conectada à entrada de interrupção, e Comunicação, na qual os pulsos de *clock* são enviados através de comandos pela rede serial.

A frequência da fonte interna pode ser configurada através da fórmula:

$$freq = \frac{60000}{1 + divisor}$$

Onde *divisor* é um número entre 1 e 65535. Assim a frequência pode variar de 0.915 Hz a 30

kHz.

A frequência da fonte externa deve ser de até 30 kHz. Seu sinal deve ser uma onda quadrada de 0 a 3.3V.

Como auxílio à execução do procedimento síncrono, é possível configurá-lo para produzir um sinal de *clock* com a mesma frequência recebida. Este sinal pode ser produzido caso haja uma placa digital conectada à PUC. Da mesma maneira, o procedimento síncrono pode ser configurado para produzir um pulso em um *bit* da placa digital após a execução do último ponto.

Caso mais de uma placa analógica ou mais de uma placa digital estejam presentes, serão utilizadas a primeira entrada analógica, a primeira saída analógica e a primeira saída digital presentes. A ordem é definida pelo valor de seus ID's, de acordo com o protocolo BSMP.

3 Entidades BSMP

Como a PUC é baseada no protocolo BSMP, a manipulação de suas funcionalidades é feita através das Entidades do protocolo. Em geral, as entidades tem ID's fixos. Entretanto, devido à natureza extensível da PUC, novas entidades são criadas caso placas de extensão estejam presentes.

As variáveis de entrada e saída analógicas e digitais são criadas de acordo com a ordem dos endereços das placas de extensão. As variáveis de entrada têm ID's menores que as respectivas variáveis de saída. Por exemplo, uma PUC com uma placa de E/S digital no endereço 0 e uma placa de E/S analógica no endereço 1 terá as seguintes variáveis de E/S (ID's 0 a 2 são fixos):

- ID 3: Entrada digital;
- ID 4: Saída digital;
- ID 5: Entrada analógica;
- ID 6: Saída analógica.

Todas as entidades da PUC são descritas nos próximos tópicos.

3.1 Variável: Placas Detectadas

ID: 0; Tamanho: 4 bytes; Somente leitura.

Contém o tipo das placas de extensão conectadas à base e identificadas pela PUC. O primeiro *byte* contém o tipo da placa no endereço 0, o segundo *byte* o tipo da placa no endereço 1, e assim por diante. Os tipos podem ser:

- 0: Placa Analógica;
- 2: Placa Digital;
- 255: Sem placa.

3.2 Variável: Estado do Procedimento Síncrono

ID: 1; Tamanho: 4 bytes; Somente leitura.

Indica o *status* atual do procedimento síncrono. O primeiro *byte* contém o estado de operação, que pode assumir os valores:

- 0: Parado;
- 1: Em execução;
- 2: Pausado.

Os três *bytes* restantes contém o índice do último ponto executado, em *big endian*. Os *bytes* podem assumir os valores:

- Mínimo: 00 00 00 → nenhum ponto executado;
- Máximo: 01 00 00 → 65536 pontos executados.

3.3 Variável: Configuração do Procedimento Síncrono

ID: 2; Tamanho: 6 bytes; Escrita.

Usada para se configurar o procedimento síncrono a ser executado.

Primeiro *byte*:

- *Bit 7*: SAÍDA → 1: Habilitada, 0: Desabilitada;
- *Bit 6*: ENTRADA → 1: Habilitada, 0: Desabilitada;
- *Bit 5*: PRECISÃO → 1: 18-bits, 0: 16-bits;
- *Bits 4 e 3*: FONTE DE CLOCK → 00: *Timer*, 01: Externo, 10: Comunicação;
- *Bits 2 a 0*: RESERVADO → escrita/leitura 000.

Segundo e terceiro *bytes*: NÚMERO DE PONTOS.

Em *big endian*. Valores válidos de 0 a 65535. O valor 0 é interpretado como 65536.

Quarto e quinto *bytes*: DIVISOR DE CLOCK.

Em *big endian*. Valores válidos de 1 a 65535.

Sexto *byte*:

- *Bit 7*: SAÍDA DE CLOCK → 1: Habilitada, 0: Desabilitada;
- *Bits 6 a 4*: BIT DE SAÍDA DE CLOCK → De 000 (*bit 0*, menos sig.) a 111 (*bit 7*);
- *Bit 3*: PULSO DE FIM → 1: Habilitado, 0: Desabilitado;
- *Bits 2 a 0*: BIT DE PULSO DE FIM → De 000 (*bit 0*, menos sig.) a 111 (*bit 7*);

3.4 Variáveis: Entradas Analógicas

Tamanho: 3 bytes; Somente leitura.

Valor lido pelo respectivo conversor A/D, em *big endian*. Valores de 00 00 00 (-10 V) a 03 FF FF (10 V).

3.5 Variáveis: Saídas Analógicas

Tamanho: 3 bytes; Escrita.

Valor a ser escrito no respectivo conversor D/A, em *big endian*. Valores de 00 00 00 (-10 V) a 03 FF FF (10V).

3.6 Variáveis: Entradas Digitais

Tamanho: 1 byte; Somente leitura.

Valor lido pela respectiva entrada digital. Valor de 00 (todos os *bits* em lógico baixo) a FF (todos os *bits* em lógico alto).

3.7 Variáveis: Saídas Digitais

Tamanho: 1 byte; Somente leitura.

Valor a ser escrito nos *bits* da respectiva saída digital. Valor de 00 (todos os *bits* em lógico baixo) a FF (todos os *bits* em lógico alto).

3.8 Curva: RAM

Tamanho do bloco: 4 kB; Número de blocos: 32; Total: 128 kB; Somente leitura.

Armazena os pontos lidos pelo procedimento síncrono. O número de pontos depende da precisão selecionada para a execução do procedimento síncrono.

3.9 Curva: Flash

Tamanho do bloco: 4 kB; Número de blocos: 32; Total: 128 kB; Escrita.

Armazena os pontos a serem executados pelo procedimento síncrono. O número de pontos depende da precisão selecionada para a execução do procedimento síncrono.

3.10 Função: Reset

Entrada: 0 bytes; Saída: 0 bytes;

Realiza um *reset* na PUC. O *reset* é imediato e equivale a um ciclo de energia na PUC. Não haverá um pacote de resposta.

3.11 Função: Iniciar Procedimento Síncrono

Entrada: 0 bytes; Saída: 0 bytes;

Inicia o procedimento síncrono. Erros possíveis:

- 01: O procedimento síncrono já estava rodando;
- 04: A configuração do procedimento síncrono estava inválida.

3.12 Função: Parar Procedimento Síncrono

Entrada: 0 bytes; Saída: 0 bytes;

Pára o procedimento síncrono. Erros possíveis:

- 03: O procedimento síncrono já estava parado.

3.13 Função: Pausar Procedimento Síncrono

Entrada: 0 bytes; Saída: 0 bytes;

Pausa o procedimento síncrono. Erros possíveis:

- 02: O procedimento síncrono já estava pausado.
- 03: O procedimento síncrono já estava parado.

3.14 Função: Passo Procedimento Síncrono

Entrada: 0 bytes; Saída: 0 bytes;

Realiza um passo do procedimento síncrono. Erros possíveis:

05: O procedimento síncrono não está em execução.

4 Biblioteca pypucsp

De modo a facilitar a operação da PUC pelos seus clientes, uma biblioteca em Python foi criada. Esta seção descreve sua API. Todos os exemplos após a Instanciação assumem uma instância de nome `p`.

4.1 Instanciação

```
SerialPUC(port, address, baud=6e6, retries=3, debug=False)
```

Cria uma instância da biblioteca. Parâmetros:

- `port` (String): nome da porta serial a ser usada.
- `address` (int): endereço configurado nos *jumpers* da PUC.
- `baud` (int): baud-rate a ser usado. Padrão: 6 Mbps.
- `retries` (int): numero de tentativas caso a transmissão de um pacote falhe. Padrão: 3 tentativas.
- `debug` (bool): habilitar a impressão de mensagens de *debug*. Padrão: não imprimir.

Exemplo: cria uma instância chamada `p` para uma PUC no endereço 8 e na porta serial `/dev/ttyUSB0`:

```
p = pucsp.SerialPUC("/dev/ttyUSB0", 8)
```

4.2 Placas detectadas

Existe uma propriedade na instância que permite ler as placas detectadas pela PUC. Essa propriedade tem o nome `detectedBoards` e pode ser obtida com:

```
p.detectedBoards
```

Esse método retorna um vetor de quatro posições, com `None` ou as *strings* “Analog” e “Digital”, correspondendo às placas presentes nos respectivos endereços. Por exemplo, se o vetor for:

```
[None, “Analog”, None, None]
```

Sabe-se que há apenas uma placa conectada à PUC, do tipo analógica e no endereço 1.

4.3 Entradas e saídas analógicas

As variáveis analógicas estão listadas em dois vetores da instância: `ads` e `das`. Caso não haja uma placa analógica conectada à base, esses vetores estarão vazios.

4.3.1 Obtendo uma referência

Para se obter uma referência para a variável analógica desejada, basta acessar o elemento correspondente do vetor `ads` ou do vetor `das`. Por exemplo: `ad` será a primeira entrada analógica, `da` será a primeira saída analógica:

```
ad, da = p.ads[0], p.das[0]
```

4.3.2 Leitura

```
read()
```

Retorna o valor de uma variável analógica (entrada ou saída). O valor retornado é um `float` entre -10.0 e 10.0. Exemplo:

```
valor_ad, valor_da = ad.read(), da.read()
```

4.3.3 Escrita

```
write(value)
```

Escreve o valor `value` em uma saída analógica. `value` deve ser um `float` entre -10.0 e 10.0. Exemplo: escreve 5.75 volts na saída analógica.

```
da.write(5.75)
```

4.4 Entradas e saídas digitais

As entradas e saídas digitais funcionam de maneira semelhante às analógicas. As listas dessas variáveis estão nas propriedades `digins` e `digouts` da instância.

4.4.1 Obtendo uma referência

Basta referenciar uma posição do vetor `digins` para entradas ou `digouts` para saída. Por exemplo: `di` será a primeira entrada digital e `do` será a primeira saída digital.

```
di, do = p.digins[0], p.digouts[0]
```

4.4.2 Leitura

```
read()
```

Retorna um valor `int` entre 0 e 255. Por exemplo:

```
valor_di, valor_do = di.read(), do.read()
```

4.4.3 Escrita

```
write(value)
```

Escreve o valor `value` em uma saída digital. `value` deve ser um `int` entre 0 e 255. Exemplo: liga os *bits* mais e menos significativos da saída digital, desligando os demais.

```
do.write(0b10000001)
```

4.4.4 Operações binárias

É possível manipular *bits* individuais das saídas digitais sem se preocupar com outros *bits*. Existem três operações. Em todas elas o parâmetro `mask` deve ser um `int` entre 0 e 255.

- `setBits(mask)`: liga os *bits* determinados por `mask`;
- `clearBits(mask)`: desliga os *bits* determinados por `mask`;
- `toggleBits(mask)`: inverte os *bits* determinados por `mask`.

Exemplo: liga o *bit* mais significativo e desliga o *bit* menos significativo; os outros *bits* não são afetados.

```
do.setBits(0b10000000)  
do.clearBits(0b00000001)
```

4.5 Reset

A função *reset* permite que a PUC seja reiniciada, como se fosse realizado um ciclo de energia. Para executá-la, basta:

```
p.reset()
```

4.6 Procedimento Síncrono

O procedimento síncrono está representado pelo objeto `sync` da instância. Através dele é possível consultar seu estado, configurá-lo, ler e escrever suas curvas e controlar sua execução.

4.7 Procedimento Síncrono: Estado

```
getState()
```

Retorna uma tupla de dois elementos com o *status* do procedimento síncrono. O primeiro elemento é uma *string* com um dos valores “RUNNING”, “STOPPED” ou “PAUSED”, indicando o estado do procedimento. O segundo elemento é um valor `int` que indica o índice do último ponto realizado, entre 0 e 65536. Exemplo:

```
print p.sync.getState() # Retorna (“STOPPED”, 0) por exemplo
```

4.8 Procedimento Síncrono: Curvas

As curvas de entrada e saída do procedimento síncrono podem ser acessadas através das propriedades `inCurve` e `outCurve`, respectivamente, do objeto `sync`.

4.8.1 Leitura

```
read(nPoints, widePoint=False)
```

Lê `nPoints` pontos da curva. O parâmetro `widePoint` indica a precisão dos pontos da curva:

- Se `widePoint == False`, os pontos têm 16 *bits* de precisão e ocupam 2 *bytes* na memória. `nPoints` pode variar de 1 a 65536;
- Se `widePoint == True`, os pontos têm 18 *bits* de precisão e ocupam 4 *bytes* na memória. `nPoints` pode variar de 1 a 32768.

É retornada uma lista de `nPoints` pontos, cada um sendo um `float` entre -10.0 e 10.0. Exemplo: lê 32000 pontos de precisão 18 *bits* da curva de entrada:

```
p.sync.inCurve.read(32000, True)
```

4.8.2 Escrita

```
write(data, widePoint=False, force=False)
```

Escreve `len(data)` pontos na curva de saída, com precisão de 16 *bits* se `widePoint` for `False` ou 18 *bits* se `True`. Caso `force` seja `True`, a curva é enviada mesmo que já exista uma curva idêntica na memória da PUC. `data` pode ter até 65536 pontos se `widePoint` for `False` ou até 32768 pontos se `widePoint` for `True`. Cada ponto deve ser um `float` de -10.0 a 10.0.

Exemplo: escreve 5 pontos de precisão 18 *bits* na memória Flash.

```
p.sync.outCurve.write([1.0, 2.0, 3.0, 4.0, 5.0], True)
```

4.9 Procedimento Síncrono: Configuração

Antes de se executar um procedimento síncrono deve-se configurar seus parâmetros. Uma classe que contém todos os parâmetros de configuração é definida na biblioteca, chamada `SyncConfig`. Os métodos de leitura e escrita de configuração retornam e aceitam objetos desta classe.

4.9.1 Classe SyncConfig

```
SyncConfig(inEnable=False, outEnable=False, widePoint=False,  
clkSource=0, nPoints=1, clkDivisor=1, clkOutEnable=False,  
clkOutBit=0, clkPulseEnable=False, clkPulseBit=0)
```


O construtor aceita todos os parâmetros definidos na seção Variável: Configuração do Procedimento Síncrono, com os mesmos limites e significados. Exemplo: criação de um objeto de configuração com apenas a saída habilitada, para 10 pontos, mantendo os outros parâmetros com seus valores padrão.

```
cfg = pucsp.SyncConfig(outEnable=True, nPoints=10)
```

4.9.2 Ler

```
getConfig()
```

Retorna um objeto da classe SyncConfig com as configurações do procedimento síncrono presentes na PUC. Exemplo:

```
cfg = p.sync.getConfig()
```

4.9.3 Escrever

```
setConfig(cfg)
```

Toma um objeto da classe SyncConfig para ajustar os parâmetros de execução do procedimento síncrono. Exemplo: configura o procedimento síncrono para entrada e saída de 500 pontos, com fonte de *clock* externa e com pontos com precisão de 18 bits.

```
cfg = pucsp.SyncConfig()
cfg.outEnable = True
cfg.inEnable = True
cfg.widePoint = True
cfg.nPoints = 500
cfg.clkSource = pucsp.SyncConfig.CLK_EXTERNAL
p.sync.setConfig(cfg)
```

4.10 Procedimento Síncrono: Execução

```
start()
stop()
pause()
step()
```

Os métodos para a execução do procedimento síncrono correspondem diretamente aos apresentados nas seções sobre as funções BSMP. Basta invocá-los diretamente. Exemplo: inicia a execução do procedimento síncrono.

```
p.sync.start()
```

5 Interface qtClient

A fim de se facilitar o teste das funcionalidades da PUC e servir como demonstração de suas capacidades, uma interface gráfica foi desenvolvida. Esta interface é mostrada na Figura 4.

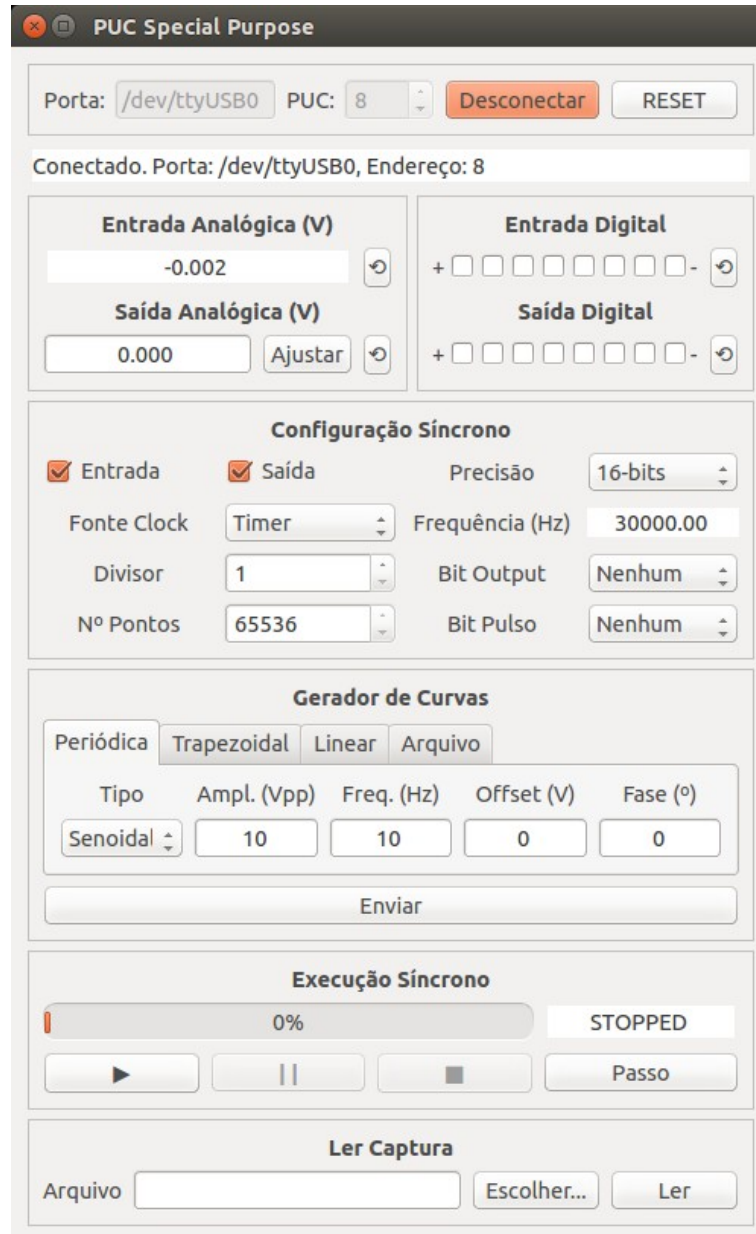


Figura 4- Interface em PyQt4

As seções subsequentes apresentam a interface em detalhes.

5.1 Conexão e Status

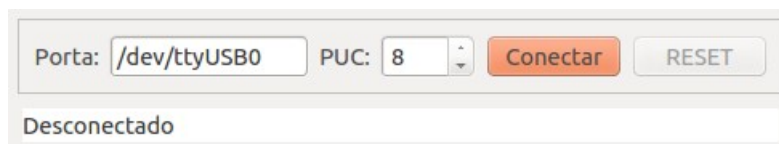


Figura 5- Interface: conexão e status

Define-se o nome da porta serial no campo “Porta” e o endereço configurado na base da PUC no campo “PUC”. Após pressionar o botão “Conectar”, se a conexão falhar, uma mensagem apropriada será escrita na barra de *status*. Se a conexão se suceder, algumas informações serão trocadas com a PUC, de modo a preencher os outros campos. O botão “Conectar” terá seu texto alterado par “Desconectar”, e o botão “RESET” será habilitado. O botão “RESET” chama a função Reset.

5.2 Escrita e leitura em variáveis



Figura 6- Interface: escrita e leitura de variáveis

Os botões com as setas circulares (*refresh*) atualizam os valores nos campos correspondentes. Os *checkboxes* da entrada digital podem ser alterados na interface, porém não emitem nenhum comando para a PUC. Eles refletem o valor lido pela entrada digital apenas após se pressionar o botão *refresh* correspondente. Já para o caso da saída digital, presionar as *checkboxes* de fato envia um comando para a PUC para que o valor seja mudado. No caso da saída analógica, o valor no campo de texto só é enviado à PUC após pressionar o botão “Ajustar”.

5.3 Configuração do procedimento síncrono

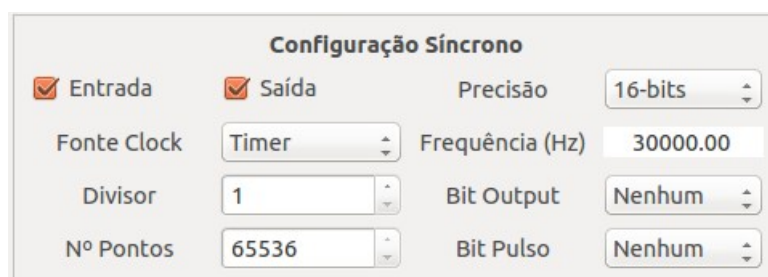


Figura 7- Interface: configuração do procedimento síncrono

Os campos de configuração correspondem às opções possíveis na PUC. O campo

“Frequência” calcula o *clock* obtido através do divisor selecionado no campo “Divisor”, caso “Fonte Clock” seja “Timer.” Caso “Fonte Clock” seja “Externo”, o campo “Frequência” torna-se editável, e deve conter a frequência externa conectada à PUC. Esse valor é utilizado na Geração de Curvas.

As alterações nos controles desse quadro só são enviadas à PUC quando o botão “Play” do quadro “Execução Síncrono” é pressionado.

5.4 Geração de curvas

O quadro “Gerador de Curvas” cria curvas com formatos selecionáveis em memória. Ao se pressionar o botão “Enviar”, a curva gerada é enviada para a PUC, sendo gravada na memória Flash. O número de pontos gerados é definido pelo campo “Nº Pontos” do quadro anterior.

Gerador de Curvas				
<input checked="" type="radio"/> Periódica <input type="radio"/> Trapezoidal <input type="radio"/> Linear <input type="radio"/> Arquivo				
Tipo	Ampl. (Vpp)	Freq. (Hz)	Offset (V)	Fase (°)
Senoidal	10	10	0	0
Enviar				

Figura 8- Interface: geração de curva periódica

Essa aba é usada para se gerar curvas periódica, que podem ser Senoidais ou Quadradas. Os valores dos pontos se baseiam no campo “Frequência” do quadro anterior para se gerar a frequência correta da curva.

Gerador de Curvas				
<input type="radio"/> Periódica <input checked="" type="radio"/> Trapezoidal <input type="radio"/> Linear <input type="radio"/> Arquivo				
Base (V)	Topo (V)	T Subida (%)	T Topo (%)	T Descida (%)
0	10	25	50	25
Enviar				

Figura 9- Interface: geração de curva trapezoidal

Essa aba gera uma curva Trapezoidal. Os valores de “Base” e “Topo” indicam o menor e o maior valor alcançados pelo trapezóide. Os campos “T Subida” e “T Topo” indicam qual a porcentagem de pontos que comporão o lado esquerdo e o lado superior do trapezóide. O campo “T Descida” é calculado automaticamente a partir dos anteriores.

Figura 10- Interface: geração de curva linear

Uma curva Linear simplesmente cria uma reta entre o a tensão de início e a tensão de fim, com o número de pontos indicados pelo campo “Nº Pontos” do quadro anterior.

Figura 11- Interface: geração de curva de arquivo

A aba “Arquivo” permite que pontos sejam lidos de um arquivo arbitrário. O arquivo deve conter um ponto por linha, sem linhas em branco entre eles e sem uma linha em branco ao fim do arquivo. Os pontos devem estar em ASCII representando um *float* entre -10.0 e 10.0.

5.5 Execução do procedimento síncrono

Figura 12- Interface: execução do procedimento síncrono

Ao se pressionar o botão “Play” a configuração do procedimento síncrono é enviada e o procedimento é iniciado (se a configuração estava correta). A barra de progresso é atualizada a cada 150 ms com o progresso da execução da curva. Se a curva está em curso, apenas os botões “Pause” e “Stop” tornam-se habilitados. Se a curva está pausada, apenas os botões “Play” e “Stop” estão habilitados. O botão Passo só é habilitado se a fonte do clock for Serial. É através dele que se dão os passos de sincronismo para o procedimento síncrono neste caso.

5.6 Leitura de curvas

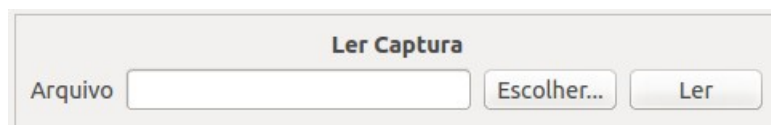


Figura 13- Interface: ler curva capturada para arquivo

Por fim, o último quadro permite ler uma curva capturada pelo procedimento síncrono. Basta “Escolher...” um arquivo (arquivos existentes serão sobrescritos!) e pressionar o botão “Ler” para que o arquivo seja preenchido com os pontos. O formato dos pontos é o mesmo descrito na geração de curvas a partir de arquivo.