Sistema de Acionamento de Válvula de Vácuo via PLC (Rockwell) e EPICS

Responsáveis: Professor Antônio Rubens

Paulo de Tarso

Patrícia Nallin

Áureo Freitas

Sumário:

1 – Introdução

O sistema de vácuo é um dos principais sistemas necessários para o funcionamento de um acelerador de partículas síncrotron. Manter o vácuo controlado é um dos principais requisitos para um bom feixe de luz síncrotron. Sirius, o novo acelerador de partículas do LNLS (Laboratório Nacional de Luz Síncrotron) de 3GeV, terá um sistema de vácuo integrado e complexo.

Conforme a Figura 1, para realizar a conexão do anel de armazenamento com as linhas de luz, a pressão externa ao anel (Pressão 1) e a pressão no sistema da linha de luz (Pressão 2), deve estar abaixo de um limite pré estabelecido, assim a integração da linha de luz com o anel poderá ocorrer.

Figura 1 – Pressões e válvula.

Para interligar os dois sistemas (anel de armazenamento e linha de luz), são utilizados dispositivos atuadores chamados válvulas. Elas permitem a interligação de vácuo dos dois sistemas.

2 – Objetivo

O objetivo do presente trabalho é realizar a operação de acionamento e controle das válvulas de vácuo. A válvula será acionada pelo PLC (Programmable Logic Controller, ou CLP – Controlador Lógico Programável em português) Rockwell Allen-Bradley da família ControlLogix 5000. Toda lógica de controle será feita em EPICS através do módulo EPICS EtherIP.

3 – Pressupostos Teóricos

Esse capítulo tem como objetivo documentar todos os tópicos teóricos necessários para entendimento e desenvolvimento de um sistema como este. Após ler e entender este presente capítulo o leitor será capaz de dar suporte ao sistema e realizar modificações no mesmo.

3.1 – PLC Rockewell – Allen Bradley

A Rockwell Automation é um fornecedor global de soluções em automação industrial, energia, controle e informação. Suas marcas em automação industrial incluem Allen-Bradley e Rockwell Software. Com sede na cidade de Milwaukee, Wisconsin, a companhia emprega cerca de 20.000 funcionários. No ano de 2017 obteve um lucro de 6.3 bilhões de dólares com crescimento de 6.3% em relação ao ano anterior [1].

A família de CPU ControlLogix 5000 possui duas entradas de ethernet. O sistema todo se comunica via rede, através do protocolo TCP/IP. Para estabelecer a conexão, pode-se destinar IPs físicos ou randômicos (DHCP) para cada uma das portas, vale notar que as portas são independentes, ou seja, podem ter dois endereços de IP diferente.

A fonte da Rockwell utilizada para alimentar o PLC, é energizada com 220V AC e tem duas saídas 24V DC que irão para o CPU e para os cartões de módulos extras. A alimentação do CPU é isolada dos cartões.

Uma CPU pode controlar uma vasta gama de módulos de diferentes tipos, como por exemplo módulo DC binário de saída, modo DC binário de entrada, modo analógico de entrada, entre muitos outros [2].

Para programar este PLC, utiliza-se a linguagem Ladder. Entretanto podemos escrever e ler as TAGs (variáveis do PLC) através da rede com o protocolo TCP/IP.

3.2 EPICS – EtherIP

Sabendo que essa família de PLC da Rockwell (ControlLogix 5000) pode, através da rede, fornecer o monitoramento e escrita de suas TAGs, Kasemir, um grande contribuinte e desenvolvedor do universo EPICS, escreveu um módulo driver em C no ano de 2001 para mapear as TAGs em variáveis EPICS [3].

Este módulo é chamado de EtherIP, sua função é basicamente transformar as TAGs em PVs EPICS. Os records das variáveis EPICS no módulo Ether/IP, poderá ser feito nos seguintes registros EPICS: ai, ao, bi, bo, mbbi, mbbo, mbbiDirect, mbboDirect [3].

O driver EtherIP comunicará com PLC através da rede com protocolo TCP/IP e ChannelAccess. Para construir seu módulo é necessário possuir um arquivo tipo “database” onde nele, será registrado as configurações de sua PV EPICS, como: tempo de scan, zero name, one name, variável de entrada ou saída, mapa de tag e outras funções. A Figura 2 mostra algumas das definições mais usuais do arquivo database (.db) [4].

Name	Summary	Description
SCAN	Scanning Algorithm	This can be one of the periodic intervals (`.1 second`, `.2 second`, `.5 second`, `1 second`, `2 second`, `5 second`, `10 second`, `I/O Intr`, `Event`, or `Passive`.
PINI	Process at Initialization	If this field is set to TRUE during database configuration, then the record is processed once at IOC initialization (before the normal scan tasks are started).
PHAS	Scan Phase Number	This field orders the records within a specific SCAN group. This is not meaningful for passive records. All records of a specified phase are processed before those with higher phase number. Whenever possible it is better to use linked passive records to enforce the order of processing rather than phase number.
EVNT	Event Number	Event number for scan type SCAN_EVENT. All records with scan type event and the same EVNT value will be processed when a call to post_event for EVNT is made. The call to post_event is: post_event(short event_number)
PRIO	Priority	Scheduling priority for processing I/O Event scanned records and asynchronous record completion tasks.
DISV	Disable Value	If DISV=DISA, then the record will be disabled, i.e. dbProcess will not process the record.
DISA	Scan Disable Input Link Value	This is the value that is compared with DISV to determine if the record is disabled. Its value is obtained via SDIS if SDIS is a database or channel access link. If SDIS is not a database or channel access link, then DISA can be set via dbPutField or dbPutLink.
SDIS	Scan Disable Input Link	An input link from which to obtain a value for DISA. This field is ignored unless it is a database link or a channel access link. If it is a database or a channel access link, dbProcess calls dbGetLink to obtain a value for DISA before deciding to call the processing routine.
PROC	Process Record	A record will be processed whenever a dbPutField is directed to this field.

Name	Summary	Description
DISS	Disable Alarm Severity	When this record is disabled, it will be put into alarm with this severity and a status of DISABLE_ALARM.
LSET	Lock Set	The lock set to which this record belongs. All records linked in any way via input, output, or forward database links belong to the same lock set. The only exception is that non-process passive input links do not force the linked record to be in the same lock set. The lock sets are determined at IOC initialization time.
LCNT	Lock Count	The number of times in succession dbProcess finds the record active, i.e. PACT is TRUE. If dbProcess finds the record active MAX_LOCK (currently set to 10) times in succession, it raises a SCAN_ALARM.
PACT	Processing Active	See Application Developers Guide for details on usage. PACT is TRUE while the record is being processed. For asynchronous records PACT can be TRUE from the time record processing is started until the asynchronous completion occurs. As long as PACT is TRUE, dbProcess will not call the record processing routine.
FLNK	Forward Link	This field is a database link. If FLNK is specified, processing this record will force a processing of the scan passive forward link record.

Figura 2 – Campos de descrição do arquivo database.

A Figura 3 demonstra uma simples implementação de um arquivo database, apenas os campos a seguir são necessários para uma básica implementação.

```
record(bo, "$(IOC):rele0")
{
        field(SCAN, "0.1 second")
        field(DTYP, "EtherIP")
        field(OUT, "@$(PLC) Saidas.Pt00.Data")
        field(ZNAM, "False")
        field(ONAM, "True")
}
```

Figura 3 – Exemplo de arquivo .db para uma PV de registro bo.

O registro “bo” significa que esta é uma PV binary output, em seguida, a string em chaves (“”) registra o nome que esta PV irá assumir. No campo SCAN é definido o tempo de atualização e busca da PV, ou seja, o tempo em que a PV será buscada, será o valor que o campo SCAN assumir, no caso 100 mili segundos. O campo STYP registra o tipo de drive utilizado neste IOC, no caso EtherIP. O campo OUT registra essa PV como uma TAG de saída do PLC, e a string entre chaves (“”) o endereço onde o IOC buscará essa TAG, no caso ele irá buscar no $PLC (variável de ambiente cadastrada no arquivo iocBoot, será abordado posteriormente) e o nome Saida.Pt00.Data o nome da TAG cadastrada na CPU do PLC. O campo ZNAM registra o valor da PV quando a mesma estiver com nível lógico baixo e o campo ONAM registra o valor da PV quando a mesma estiver com nível lógico alto.

```
#!/opt/epics-R3.14.12.7/modules/ether_ip-ether_ip-2-27/bin/linux-x86_64/eipIoc
# 3.14 example startup file for a Host  -*- shell-script -*-

dbLoadDatabase("../../dbd/eipIoc.dbd")
eipIoc_registerRecordDeviceDriver(pdbbase)

epicsEnvSet("EPICS_IOC_LOG_INET", "127.0.0.1")
epicsEnvSet("EPICS_IOC_LOG_PORT", "6505")
epicsEnvSet("EPICS_CAS_SERVER_PORT", "5100")
epicsEnvSet("EPICS_CAS_INTF_ADDR_LIST", "10.0.6.75")
EIP_buffer_limit(450)
drvEtherIP_init()

EIP_verbosity(7)

drvEtherIP_define_PLC("plc1", "10.0.28.140", 0)

dbLoadRecords("../../db/vacuo.db", "PLC=plc1,IOC=VAC")

iocInit()
```

Figura 4 – Exemplo de arquivo .cmd.

Um outro arquivo indispensável para construção de um IOC utilizando o módulo EtherIP é o arquivo .cmd. Este arquivo é responsável pela conexão com PLC e com a base EPICS. A Figura 4 demonstra a implementação simples de um arquivo .cmd.

Neste arquivo podemos ver os principais elementos de um arquivo .cmd. Primeiramente é carregado os arquivos .dbd utilizados pelo módulo EPICS, “dbLoadDatabase” com a função do driver EtherIP e com a função “eipIoc_registerRecordDeviceDriver”. Esse arquivo também é responsável por setar as variáveis de ambiente responsáveis para aplicações em todo o IOC, com o comando “epicsEnvSet” . Com o comando “drvEtherIP_define_PLC” definimos o nome, o endereço de rede do PLC Rockwell a ser utilizado e em qual CPU do sistema iremos procurar as TAGs. Com o comando “dbLoadRecords” carregamos o nosso arquivo .db para construir as nossas PVs EPICS e finalmente com o comando “iocInit()” startamos o IOC do módulo EtherIP.

De acordo com Kasemir, no GitHub oficial do EtherIP [5], este driver apenas funcionará com a base EPICS 3.14. O autor deste trabalho realizou o teste construir o EtherIP com a base EPICS 3.15 e não obteve sucesso, confirmando a abordagem de Kasemir.

No link https://github.com/aureocarneiro/valvula_prof_rubens., têm uma série de registros de PVs em arquivo database para construir todos os tipos de PVs: ai, ao, bi, bo, mbbi, mbbo, mbbiDirect, mbboDirect.

4 – Metodologia

Primeiramente foram cadastradas TAGs a serem utilizadas no PLC através do programa da Rockwell. As TAGs cadastradas foram divididas entre entradas, saída e memórias. Abaixo segue a lista das TAGs cadastradas.

- Saídas: Saidas.Pt00.Data, Saidas.Pt01.Data, Saidas.Pt02.Data, Saidas.Pt03.Data.
- Entradas: Entradas.Pt00.Data, Entradas.Pt01.Data, Entradas.Pt04.Data, Entradas.Pt05.Data.
- Memórias: Memoria_00, Memoria_01, Memoria_02, Memoria_03, Memoria_04, Memoria_05.

As memórias foram cadastradas devido a impossibilidade de escrever diretamente em uma entrada ou saída através da rede, ou seja, para acionar remotamente uma saída do PLC é preciso linkar uma memória a ela (No código embarcado, em Ladder, no PLC) e acionar a memória, ai sim pode-se escrever na saída, exemplificando, toda vez que a memória assumir nível lógico alto a saída

também assumirá e o contrário também é verdadeiro. Na Figura 5 é demonstrado como foi feito no Ladder do PLC esse Link de uma memória com a saída.

Figura 5 – Link de memória com a TAG de saída.

Para o acionamento da válvula, a largura de pulso do sinal deverá ser de no mímimo 50ms. Assim, nas TAGs de saída Saidas.Pt00.Data e Saidas.Pt01.Data foi adicionado um Timer conforme demonstrado na Figura 6. Esse Timer possui largura de pulso de 100ms. É possível associar à TAG Timers[0] a uma variável EPICS e conforme o valor que a variável EPICS assumir com o comando “caput”, o valor desse Timer[0] será modificado.

Figura 6 – Lógica de temporização embarcada no PLC.

Um outro passo importante é a configuração da porta ethernet. A configuração é feita no software da Rockwell. Para essa aplicação foi configurado um IP estático com endereço “10.0.28.140” com máscara de “255.255.255.0” na porta A do PLC. A comunicação do IOC com o PLC foi feita com conexão ponto-a-ponto.

O setup do PLC utilizado foi: Cartão Fonte DC 24V, CPU ControlLogix família 5000, Cartão DC Input e cartão DC Output.

Após toda lógica Ladder ser feita, todas as TAGs serem cadastradas no PLC, a configuração de IP estabelecida e o circuito do PLC montado, foi feito um arquivo .db associando as TAGs de entrada, saída e memória a variáveis EPICS. O arquivo em anexo a este documento apresenta o arquivo .db utilizado no IOC. O nome do arquivo é “vacuo.db”.

Após a configuração do arquivo .db, foi construído o arquivo .cmd. O arquivo .cmd foi nomeado de “st.cmd.host” e está em anexo a este documento. Cada variável de ambiente configurada no arquivo possui uma explicação. No capítulo de referências deste documento existe um link, tópico 6, com o documento explicando cada uma delas [6].

Na função drvEtherIP_define_PLC é passado o IP configurado previamente do PLC e o módulo da CPU, no caso o módulo é 0 pois só possuímos uma única CPU no sistema elaborado. O nome plc1 é o nome onde é mascarado o IP e o módulo para associações com o arquivo .db.

Na função dbLoadRecords é configurado o arquivo .db do sistema e é passado para ele as variáveis de ambiente PLC e IOC. A variável de ambiente PLC é onde o arquivo .db buscará as TAGs do PLC e o arquivo IOC é o nome dos prefixos que as PVs irão assumir.

Após o arquivo .cmd estar configurado basta executar o IOC do sistema, assim todas as PVs EPICS estarão online. Para acessar as variáveis EPICS basta configurar as variáveis de ambiente EPICS_CA_ADDR_LIST para o IP onde o IOC EtherIP está rodando, assim é possível acessar as PVs EPICS de qualquer ponto que estaja na mesma rede onde o IOC está rodando.

Ao acessar as variáveis EPICS é possível escrever um programa em Python com o módulo pyepics para monitorar essas PVs e escrever nelas [7]. O único requisito é que a máquina que executará o programa Python esteja na mesma rede que o IOC Ether/IP. Encontra-se em anexo a este documento um programa em Python simples para ler e escrever nessas PVs, assim fazendo a lógica de controle do PLC em alto nível, não sendo necessário escrever o programa em Ladder.

5 – Resultados

Com tudo configurado e funcionando foi feito um software em Java, com a IDE cs-studio para realizar o monitoramento e simulação do IOC [8]. A Figura 7 mostra a tela principal do software.

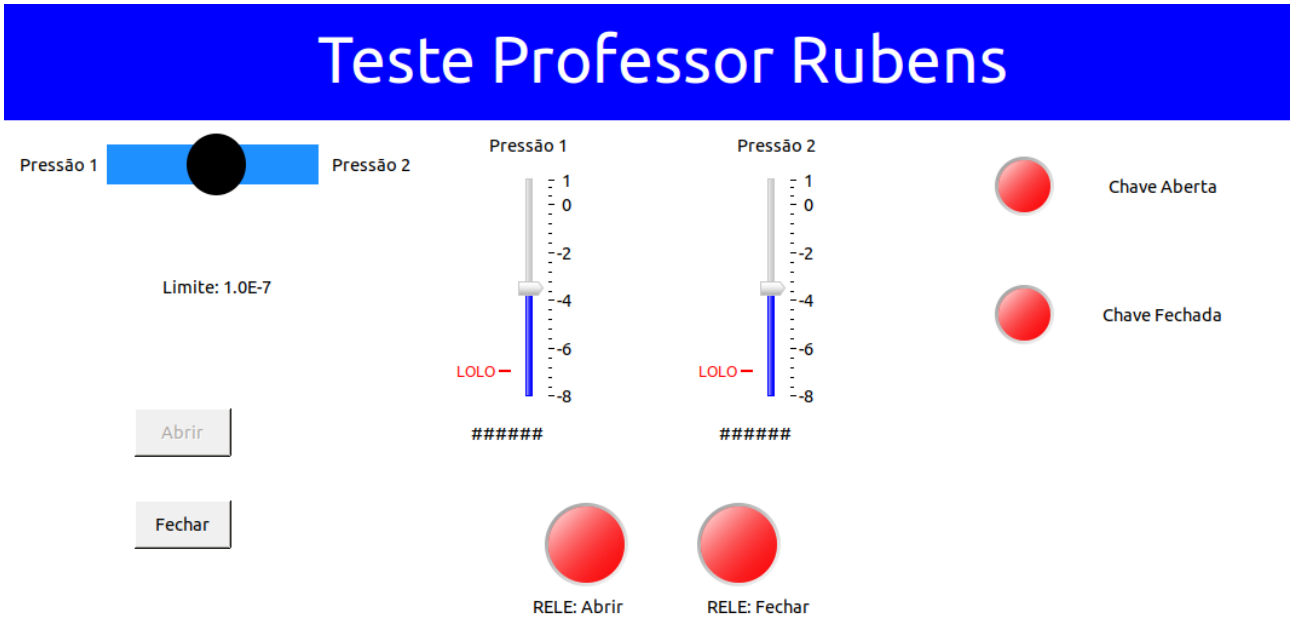

Figura 7 – Tela principal do software.

A especificação do sistema foi feita pelo Prof. Rubens. O software basicamente monitora as PVs do sistema e muda sua configuração de acordo com a leitura das PVs, por exemplo, o botão para abrir a válvula só estará habilitado caso as pressões 1 e 2 estejam abaixo do limite estabelecido e o fechamento da válvula poderá ser feito a qualquer momento.

O sistema foi testado no dia 05 de outubro de 2018 com a presença de todos os autores do projeto. Foi válidado através de um teste prático mas ainda não está implementado.

6 – Conclusão

O sistema obteve um tempo de resposta satisfatório (em torno de 100 ms), baseando no fato de ter uma lógica rodando em alto nível. Escrever um programa em alto nível para um PLC é uma tarefa muito complicada, entretanto, com esse sistema a dificuldade dessa tarefa tornou-se mínima.

Foi feito um repositório no GitHub contendo todos os arquivos utilizados no sistema.

Link: https://github.com/aureocarneiro/valvula_prof_rubens.

Referências

[1] - https://www.rockwellautomation.com/global/about-us/overview.page?pagetitle=Our-Company&docid=d56238b9aed19ae573610a1eed5cd5ed

[2] - https://ab.rockwellautomation.com/pt/IO/Chassis-Based

[3] - http://www.slac.stanford.edu/econf/C011127/talks/THDT002.pdf

[4] - https://epics.anl.gov/EpicsDocumentation/AppDevManuals/RecordRef/Recordref.pdf

[5] - https://github.com/EPICSTools/etherip

[6] - https://epics.anl.gov/base/R3-14/10-docs/CAref.html#EPICS

[7] - http://cars9.uchicago.edu/software/python/pyepics3/overview.html

[8] - https://github.com/lnls-sirius/cs-studio

ANEXO – Arquivo vacuo.db

```
record(bo, "$(IOC):rele0")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(OUT, "@$(PLC) Saidas.Pt00.Data")
    field(ZNAM, "False")
    field(ONAM, "True")
}
record(bo, "$(IOC):rele1")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(OUT, "@$(PLC) Saidas.Pt01.Data")
    field(ZNAM, "False")
    field(ONAM, "True")
}
record(bo, "$(IOC):rele2")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(OUT, "@$(PLC) Saidas.Pt02.Data")
    field(ZNAM, "False")
    field(ONAM, "True")
}
record(bo, "$(IOC):rele3")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(OUT, "@$(PLC) Saidas.Pt03.Data")
    field(ZNAM, "False")
    field(ONAM, "True")
}
record(bi, "$(IOC):entrada0")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(INP, "@$(PLC) Entradas.Pt00.Data")
    field(ZNAM, "False")
    field(ONAM, "True")
}
```

```
record(bi, "$(IOC):entrada1")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(INP, "@$(PLC) Entradas.Pt01.Data")
    field(ZNAM, "False")
    field(ONAM, "True")
}
record(bi, "$(IOC):entrada4")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(INP, "@$(PLC) Entradas.Pt04.Data")
    field(ZNAM, "False")
    field(ONAM, "True")
}
record(bi, "$(IOC):entrada5")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(INP, "@$(PLC) Entradas.Pt05.Data")
    field(ZNAM, "False")
    field(ONAM, "True")
}
record(bo, "$(IOC):memoria0")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(OUT, "@$(PLC) Memoria_00")
    field(ZNAM, "False")
    field(ONAM, "True")
}
record(bo, "$(IOC):memoria1")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(OUT, "@$(PLC) Memoria_01")
    field(ZNAM, "False")
    field(ONAM, "True")
}
```

```
record(bo, "$(IOC):memoria2")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(OUT, "@$(PLC) Memoria_02")
    field(ZNAM, "False")
    field(ONAM, "True")
}
record(bo, "$(IOC):memoria3")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(OUT, "@$(PLC) Memoria_03")
    field(ZNAM, "False")
    field(ONAM, "True")
}
record(bo, "$(IOC):memoria4")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(OUT, "@$(PLC) Memoria_04")
    field(ZNAM, "False")
    field(ONAM, "True")
}
record(bo, "$(IOC):memoria5")
{
    field(SCAN, "0.1 second")
    field(DTYP, "EtherIP")
    field(OUT, "@$(PLC) Memoria_05")
    field(ZNAM, "False")
    field(ONAM, "True")
}
```

ANEXO – Arquivo st.cmd.host

```
#!/opt/epics-R3.14.12.7/modules/ether_ip-ether_ip-2-27/bin/linux-x86_64/eipIoc
# 3.14 example startup file for a Host  -*- shell-script -*-

dbLoadDatabase("../../dbd/eipIoc.dbd")
eipIoc_registerRecordDeviceDriver(pdbbase)

epicsEnvSet("EPICS_IOC_LOG_INET", "127.0.0.1")
epicsEnvSet("EPICS_IOC_LOG_PORT", "6505")
epicsEnvSet("EPICS_CAS_SERVER_PORT", "5100")
epicsEnvSet("EPICS_CAS_INTF_ADDR_LIST", "10.0.6.75")
EIP_buffer_limit(450)
drvEtherIP_init()

EIP_verbosity(7)

drvEtherIP_define_PLC("plc1", "10.0.28.140", 0)

dbLoadRecords("../../db/vacuo.db", "PLC=plc1,IOC=VAC")

iocInit()
```

ANEXO – Arquivo Python para leitura e set de PVs EPICS

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# importando os módulos necessários
from epics import PV, caput
import time
import sys

# criando so objetos que irão monitorar as PVs de pressão
leit_1 = PV('VAC:int1-RB')
leit_2 = PV('VAC:int2-RB')

# criando os objetos de pv de acionamento das válvulas
acio_1 = PV('VAC:rele0')
acio_2 = PV('VAC:rele1')

# criando os objetos de pv de chaves de fim de curso
chave_open = PV('VAC:memoria4')
chave_close = PV('VAC:memoria5')

# definindo o limite no nível de vácuo
limite = -7

# definindo variáveis de PV
rele0 = 'VAC:rele0'
rele1 = 'VAC:rele1'
rele2 = 'VAC:rele2'
rele3 = 'VAC:rele3'
pressao = 'VAC:int1-RB'
pressao = 'VAC:int2-RB'
memoria0 = 'VAC:memoria0'
memoria1 = 'VAC:memoria1'
memoria2 = 'VAC:memoria2'
memoria3 = 'VAC:memoria3'
memoria4 = 'VAC:memoria4'
memoria5 = 'VAC:memoria5'
entrada0 = 'VAC:entrada0'
entrada1 = 'VAC:entrada1'
entrada4 = 'VAC:entrada4'
entrada5 = 'VAC:entrada5'
```

```
# definindo sinais
up = 1
down = 0

# definindo tempo de duty cicle
dutycycle = 0.5

# loop principal
while (1):

    # pré configuração inicial
    state0 = 0
    state1 = 0
    caput('VAC:memoria0', 0)
    caput('VAC:memoria1', 0)

    # estado de abertura da valvula
    while(leit_1.value <= limite and leit_2.value <= limite and chave_open.value == 1):
        # acoes a serem tomadas no estado
        state0 = 1
        state1 = 0
        #caput(memoria0,state0)
        #caput(memoria1,state1)


    # estado de fechamento da valvula
    while(leit_1.value > limite or leit_2.value > limite and chave_open.value == 1):
        # acoes a serem tomadas no estado
        caput('VAC:memoria1',1)
        caput('VAC:memoria0',0)
```