

FEUPAutom

Este programa permite desenvolver programas numa linguagem chamada ST (Structured Text, texto estruturado) e controlar uma simulação e/ou comandar hardware real.

Este projecto foi desenvolvido por Armando Jorge Sousa e Paulo Costa, também com contribuições de José António Faria, do DEEC da FEUP, Portugal.

Data da última actualização deste ficheiro: Sábado, 28 de Outubro de 2006

Armando Jorge Sousa – asousa@fe.up.pt



Índice

FEUPAutom.....	1
Índice.....	2
1.Ambiente de Desenvolvimento do FEUPAutom.....	3
2.Resumo das funcionalidades de edição.....	5
3.Introdução ao FEUPAutom e ao Texto Estruturado (ST).....	6
3.1.Estado das entradas e saídas.....	6
4.O Ciclo do FEUPAutom.....	8
5.Execução.....	8
6.Simulações.....	8
7.Disposição Sugerida.....	9
8.Tempo e Temporizadores.....	9
9.Traçados Temporais (log).....	9
10.Máquinas de Estados.....	10
11.Implementação manual de Grafcet.....	10
12.Comunicação ModBus.....	10
13.Exemplos de Programação.....	10
13.1.Trabalhando com bits.....	10
13.2.Trabalhando com Inteiros (Words).....	11
13.3.Trabalhando com bits e words.....	11
13.4.Int To Bin – testa todas as combinações de saída.....	12
13.5.Trabalhando com tempo real	12
13.6.Trabalhando com Temporizadores.....	13
13.7.Temporizador com TOn e TOff controlável.....	13
13.8.Máquina de Estados – Estados são bits com nome descritivo.....	14
13.9.Máquina de estados – estado num inteiro.....	15
13.10.Máquinas de estados com temporizadores.....	16
13.11.Operações ao flanco ascendente e descendente.....	17
13.12.Temporizadores do Utilizador.....	17
13.13.Máquinas de estado e Temporizadores do Utilizador.....	17
13.14.Máquinas de estado e Temporizadores (exemplo melhorado).....	18
14.Dicas para programação em ST.....	19
15.Situações conhecidas e como as contornar.....	19
16.Funcionalidade dos bits do sistema.....	20
17.Referência de ST.....	21

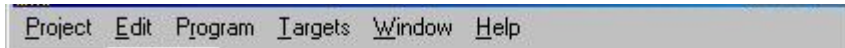
1. Ambiente de Desenvolvimento do FEUPAutom

A janela principal do FEUPAutom contém:

- Título da janela incluindo versão do FEUPAutom e o título do projecto



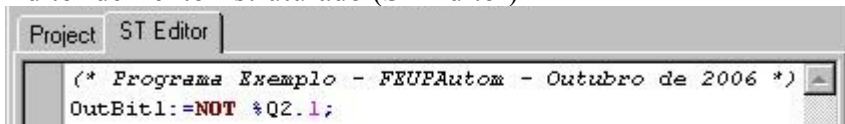
- Barra de menus:



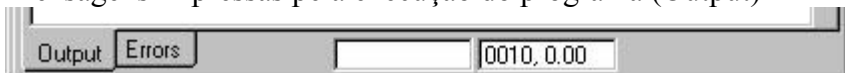
- Editor de projecto contendo autor, etc (Project)



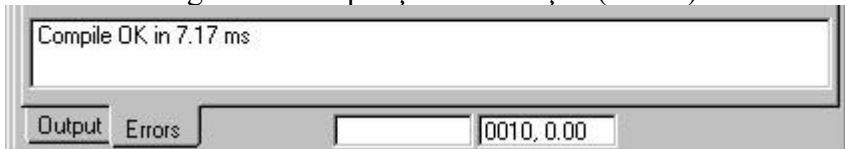
- Editor de Texto Estruturado (ST Editor)



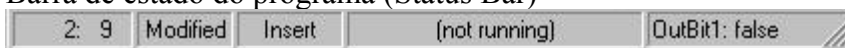
- Mensagens impressas pela execução do programa (Output)



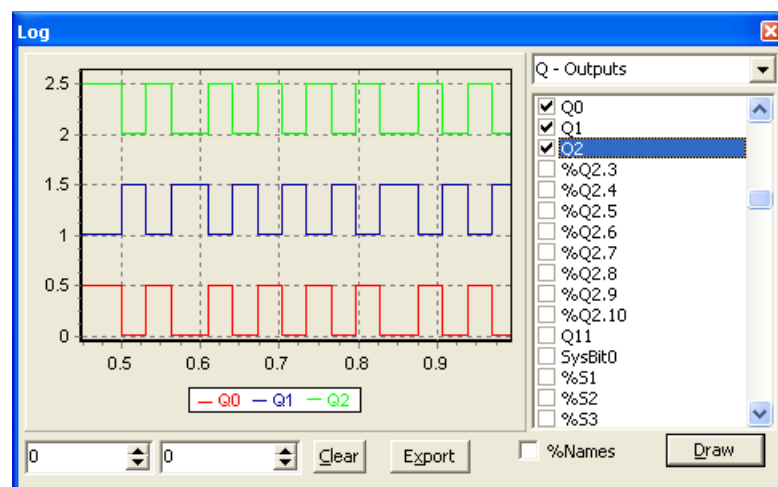
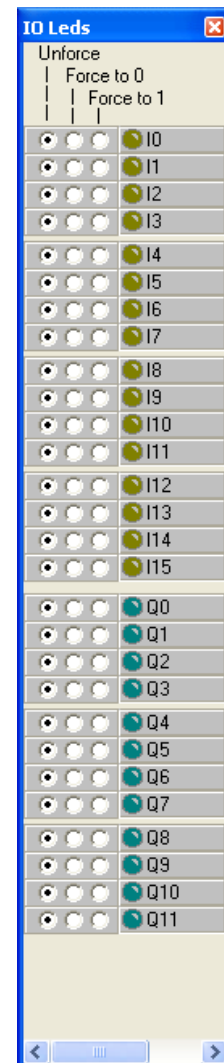
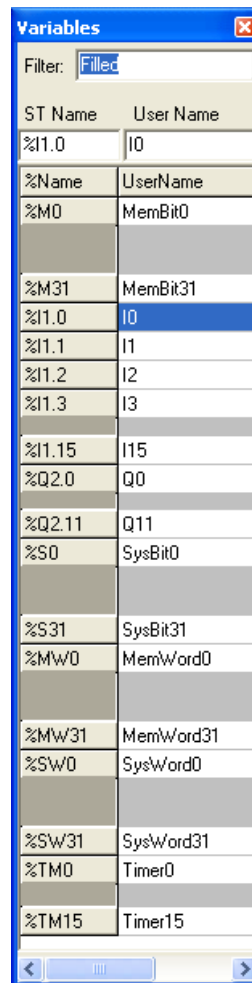
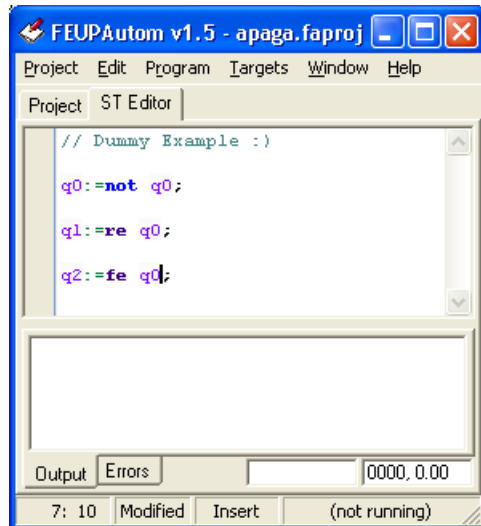
- Erros e mensagens de compilação e execução (Errors)



- Barra de estado do programa (Status Bar)



- Aspecto Geral, diversas janelas:



2. Resumo das funcionalidades de edição

- A divisão entre a zona das mensagens e a zona de edição é alterável
- A barra de estado é alterada frequentemente e se o cursor do rato estiver sobre uma variável, o seu valor será mostrado
- Clicando sobre uma mensagem de erro o cursor de edição é levado até ao ponto onde o compilador determinou a existência do erro
- Os comandos normais de edição de e para o *clipboard* estão disponíveis (copy CTRL+C, cut CTRL+X, paste CTRL+V); também o undo CTRL Z está disponível.
- Durante a execução do script, não é possível alterá-lo!
- É possível definir marcadores com Shift+Ctrl+número; para chamar premir Ctrl+número
- Utilizar CTRL+SPACE para chamar o menu de completação automática de código (“completion”)
- As variáveis e as keywords são reconhecidas pelo editor e colorizadas em conformidade (“Syntax highlighting”)

3. Introdução ao FEUPAutom e ao Texto Estruturado (ST)

O controlador FEUPAutom lê “entradas” e comanda “saídas”. Estas entradas e saídas digitais só podem ter o estado TRUE ou FALSE.

À falta de definição em contrário, os nomes das variáveis de entrada e saída são I0 a I15 e Q0 a Q15. Por motivos tecnológicos, as entradas são também designadas por %I1.x onde x vai desde 0 até 15. Dentro desta lógica, quando se refere por exemplo a entrada 3, está a ser mencionado o bit %I1.3. Dentro da mesma lógica, as saídas são designadas por %Q2.x, onde x vai desde 0 até 11.

Não é possível definir novas variáveis não previstas.

Todas as variáveis que existem estão listadas na janela Variables (menu Window -> Variables) – ver figura ao lado.

É possível utilizar variáveis auxiliares de memória que podem ser bits (variáveis booleanas binárias) ou inteiras.

Os bits auxiliares de memória são designados por MemBitx e por %Mx, onde x vai desde 0 a 31.

As memória auxiliares inteiras (words) são designados por MemWordx e por %MWx, onde x vai desde 0 a 31.

Existem ainda bits e inteiros especiais do sistema SysBitx (%Sx) e SysWordx (%SWx).

Relativamente às variáveis, é importante reter as seguintes noções:

- É possível dar nomes definidos pelo utilizador a cada variável
- Todas as variáveis são globais
- Todas as variáveis têm de estar mapeadas em memória, no sistema ou em entradas/saídas
- Todas as variáveis inteiras são inicializadas a 0
- Todos os bits são inicializados a FALSE
- Clear Memory, Clear IOs e Clear System que repõem a 0 e FALSE todas as memórias e os bits de entrada/saída
- Não pode haver nomes de variáveis repetidas
- Os nomes das variáveis podem conter apenas letras, números e '_' – NÃO utilizar caracteres portugueses nem espaços nem pontos!!!
- As variáveis não distinguem maiúsculas e minúsculas (não são 'case sensitive')
- Não dar nomes reservados tal como: for, to, do, if, then, else, elsif, true, false, RE, FE, etc.

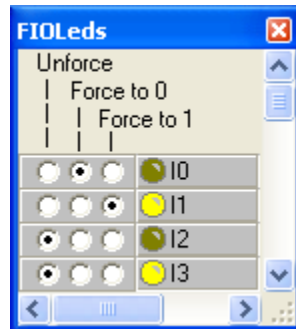
ST Name	User Name
%TM6	
%Name	UserName
%M0	MemBit0
%M31	MemBit31
%I1.0	I0
%I1.15	I15
%Q2.0	Q0
%Q2.11	Q11
%S0	SysBit0
%S31	SysBit31
%MW0	MemWord0
%MW31	MemWord31
%SW0	SysWord0
%SW31	SysWord31
%TM0	Timer0
%TM15	Timer15

3.1. Estado das entradas e saídas

É possível examinar os estados das entradas e saídas (IOs) através da janela respectiva Window->IOLEds.

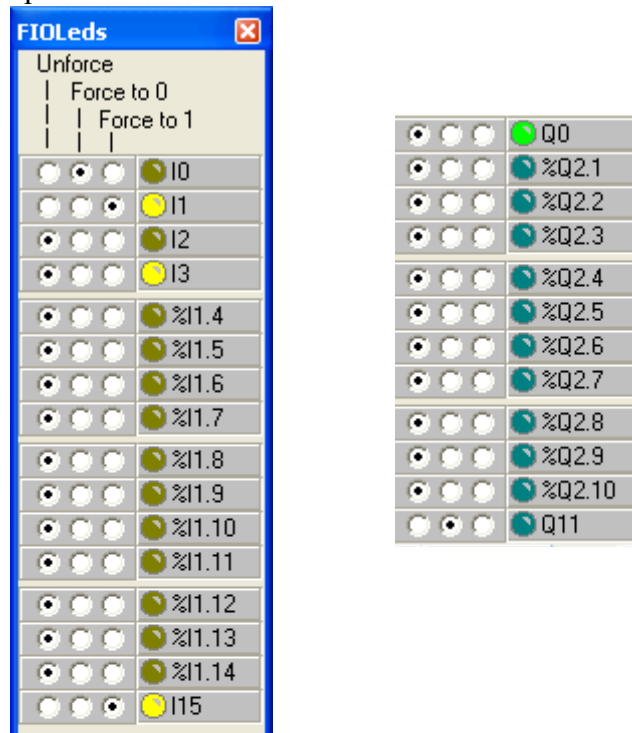
Os “leds” acendem quando um bit de IO fica activo. É possível forçar determinado bit a TRUE ou FALSE se se seleccionar a Check Box respectiva. Depois de a check box estar seleccionada, este bit é “forçado” a um determinado estado. Premir o led para alternar o estado do bit em causa, permitindo desta forma forçar um bit a TRUE ou a FALSE. Esta técnica funciona para tanto para entradas como para saídas.

No seguinte exemplo:



- Bit de entrada 0, I0, está forçado a 0 (FALSE).
- Bit de entrada 1, I1, está forçado a TRUE
- Bit de entrada 2, I2, está a FALSE (este bit não está forçado, este é o valor que a simulação determinou para esta entrada)
- Bit de entrada 3, I3, está a TRUE (este bit não está forçado, este é o valor que a simulação determinou para esta entrada)

É possível fazer o mesmo para as saídas.



4. O Ciclo do FEUPAutom

O código que o autor do projecto escreve denomina-se de Script.

O Script funciona em ciclo infinito.

O ciclo de funcionamento é sempre o seguinte:

- Leitura de entradas para imagens das entradas (incluindo comunicação com a simulação)
- Execução do script do utilizador sobre imagens das entradas e imagens das saídas
- Escrita das imagens das saídas para as saídas (incluindo comunicação com a simulação)
- Execução de tarefas de sistema (cálculos relacionados com temporizadores, etc)

O FEUPAutom não permite a existência de ciclos prolongados dentro do script uma vez que isso impediria a leitura e escritas das entradas e saídas do controlador.

Ciclos infinitos causam um timeout que é assinalado na barra de estado da aplicação.

5. Execução

Existem 2 modos de execução, Run – execução em ciclo e Run Once – execução um ciclo apenas.

- Apenas no modo de Run Once, é possível visualizar as mensagens impressas pelo programa. Esta funcionalidade não é do padrão ST. Para imprimir variáveis utilizar os comandos `print(variavel)` e `println(variavel)`. É possível imprimir 'mensagens' entre 'plicas' (não "aspas"). Os resultados aparecem no separador “Output” (parte inferior do editor)
- Em qualquer altura é possível investigar um valor de uma variável passando o rato por cima do username da variável em questão (não funciona para 'percentnames').
- A funcionalidade anterior permanece actualizada mesmo durante o running do script desde que o rato se movimente sempre

6. Simulações

O FEUPAutom pode funcionar associado a uma simulação (menu Target).

A simulação é um programa separado.

À data deste documento, as simulações disponíveis são o 3D_Park e o 3D_Gate.

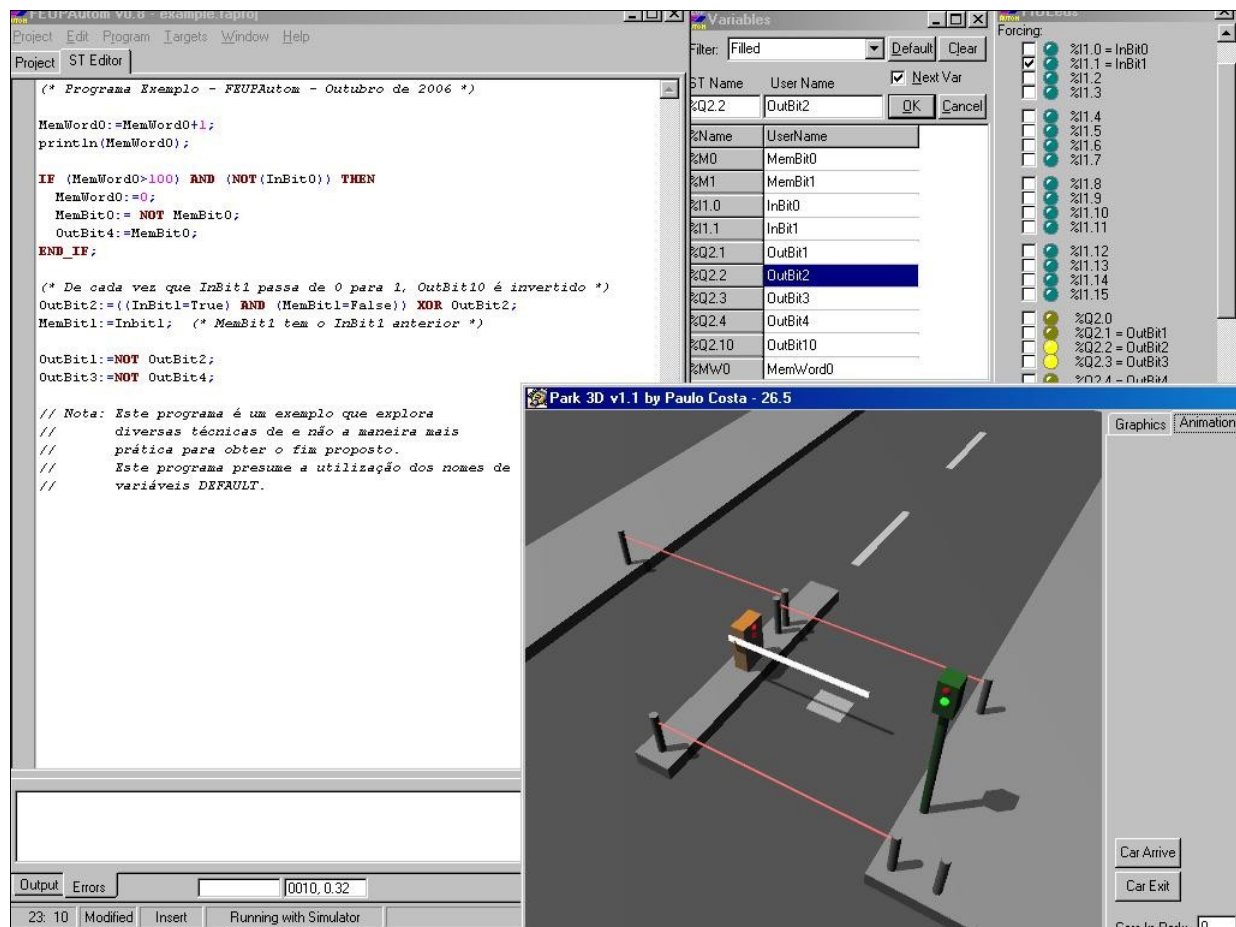
Quando a simulação está a funcionar e o FEUPAutom está RUNNING, determinados bits de saída do FEUPAutom comandam a simulação e a simulação comanda as entradas do FEUPAutom. A afirmação anterior presume que os bits não estejam forçados (nem no FEUPAutom nem na simulação).

A documentação do que cada bit representa está na guião do trabalho. As ligações entre FEUPAutom e a Simulação não são configuráveis.

Os directórios do FEUPAutom e da simulação devem estar ao mesmo nível da árvore de directórios. É equivalente arrancar a simulação dentro do FEUPAutom ou fora (no Sistema Operativo).

7. Disposição Sugerida

Para uma utilização mais agradável pode ser interessante subir a resolução do monitor (se possível).



8. Tempo e Temporizadores

(Por favor consultar o exemplo de programação “Temporizadores”)

9. Traçados Temporais (log)

(não escrito)

Limitações:

- SysBit0 e SysBit16 não guardados
- SysWord0 e SysWord16 não guardados

10. Máquinas de Estados

(por favor consultar os exemplos de programação)

11. Implementação manual de Grafcet

(por favor consultar os exemplos de programação)

12. Comunicação ModBus

(não implementado)

13. Exemplos de Programação

13.1. Trabalhando com bits

```
(*****)  
(* PrintBit.faproj - Imprime o tempo real de programa *)  
(* Testar premindo F8 repetidamente *)  
(*****)  
OutBit1:=InBit0;      // em cada ciclo, copia para a saída 1 a entrada 0  
OutBit2:=NOT InBit0;  // em cada ciclo, copia para a saída 1 a entrada 0 negada  
println(InBit0);      // imprime 0 para False e -1 para True
```

13.2. Trabalhando com Inteiros (Words)

```
(*****)  
(* PrintWords.faproj - Exemplo de impressão de inteiros *)  
(* Testar premindo F8 repetidamente *)  
(*****)  
  
IF MemWord0=0 THEN  
    MemWord1:=1000;  
END_IF;  
  
print('Inteiro de memória 0=');  
print(MemWord0);  
print('; Inteiro de memória 1=');  
println(MemWord1);  
print('O dobro valor no inteiro de memória 1=');  
println(MemWord1*2);  
  
MemWord0:=MemWord0+1;  
MemWord1:=MemWord1+1;
```

13.3. Trabalhando com bits e words

```
(*****)  
(* Example.faproj - exemplo com bits e inteiros *)  
(*****)  
(* Este programa é apenas um exemplo exploratório *)  
(*****)  
MemWord0:=MemWord0+1;  
println(MemWord0);  
IF (MemWord0>100) AND (NOT(InBit0)) THEN  
    MemWord0:=0;  
    MemBit0:= NOT MemBit0;  
    OutBit4:=MemBit0;  
END_IF;  
(* De cada vez que InBit1 passa de 0 para 1, OutBit10 é invertido *)  
OutBit2:=((InBit1=True) AND (MemBit1=False)) XOR OutBit2;  
MemBit1:=Inbit1; (* MemBit1 tem o InBit1 anterior *)  
OutBit1:=NOT OutBit2;  
OutBit3:=NOT OutBit4;
```

13.4. Int To Bin – testa todas as combinações de saída

```
(*****)
(* IntToBin.faproj - Testa todas as combinações de saídas *)
(*****)

Delay:=32;
AllOutputs:=2*2*2*2*2*2*2*2*2*2*2*2; // (12 Outputs)=>2^12

Contador:=(Contador+1) mod (AllOutputs*Delay);

OutBit0:= ((Contador/Delay) AND 1) <>0;
OutBit1:= ((Contador/Delay) AND 2) <>0;
OutBit2:= ((Contador/Delay) AND 4) <>0;
OutBit3:= ((Contador/Delay) AND 8) <>0;
OutBit4:= ((Contador/Delay) AND 16) <>0;
OutBit5:= ((Contador/Delay) AND 32) <>0;
OutBit6:= ((Contador/Delay) AND 64) <>0;
OutBit7:= ((Contador/Delay) AND 128) <>0;
OutBit8:= ((Contador/Delay) AND 256) <>0;
OutBit9:= ((Contador/Delay) AND 512) <>0;
OutBit10:= ((Contador/Delay) AND 1024) <>0;
OutBit11:= ((Contador/Delay) AND 2048) <>0;
```

13.5. Trabalhando com tempo real

```
(*****)
(* Print.faproj - Imprime o tempo real de programa *)
(* Testar premindo F8 repetidamente *)
(*****)

print('Tempo real, em segundos, desde o inicio programa=');

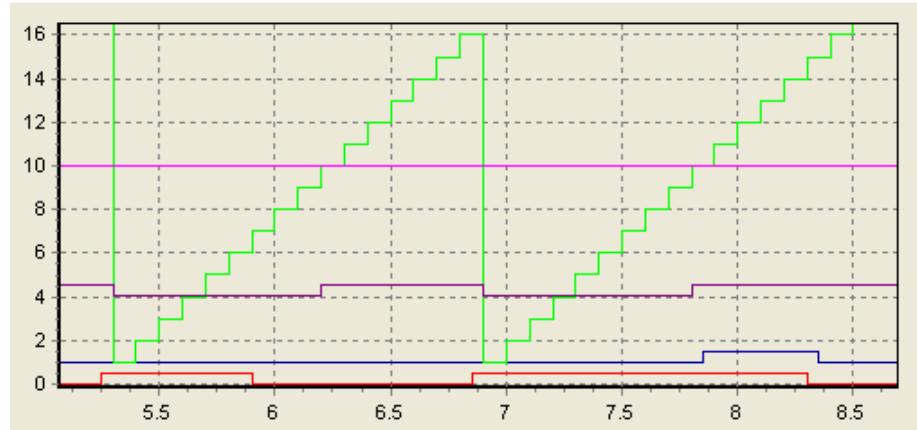
println(SysWord16/10);
```

13.6. Temporizador Simples TOn

```
(*****)
(* Timer_TOn_Simple.faproj - Exemplo de Timers *)
(* Modo TOn: Liga após xxx.P *)
(*****)
```

```
if RE i0 then
  t0.mode:=ton;
  t0.p:=10;
  start t0;
end_if;

q0:=i0 and t0.q;
```

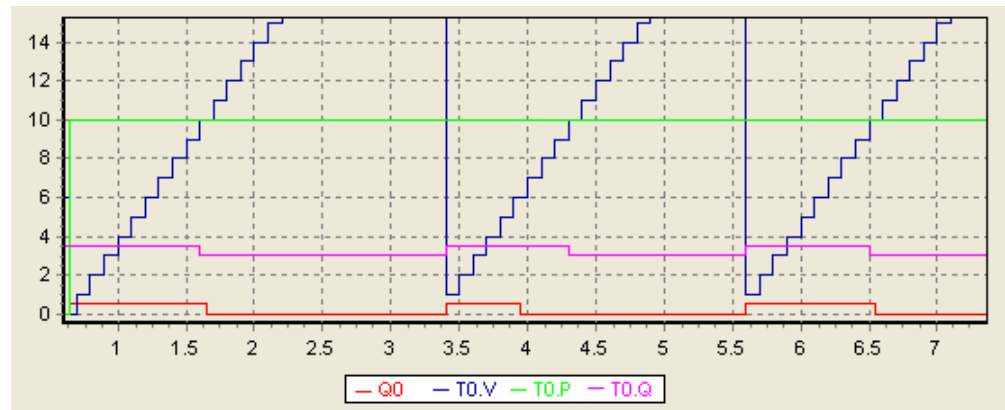


13.7. Temporizador Simples TOff

```
(*****)
(* Timer_TOff_Simple.faproj - Exemplo de Timers *)
(* Modo TOff: Liga logo e desliga após xxx.P décimos de seg *)
(*****)
```

```
if RE i0 then
  t0.mode:=toff;
  t0.p:=10;
  start t0;
end_if;

q0:=i0 and t0.q;
```



13.8. Temporizadores

```
(*****)
(* TimerSimple2.faproj - Exemplos de Timers *)
(* Modo TOn: Liga após xxx.P *)
(* Modo TOff: Liga logo e desliga após xxx.P décimos de seg *)
(* Pisca Q0 1 seg on e 1 seg off *)
(*****)

// Obs: a word de sistema 0 conta o número de ciclos executados
// e pode ser utilizada para inicializações

// IMPORTANTE: para repor sw0:=0, premir CTRL+SHIFT+R

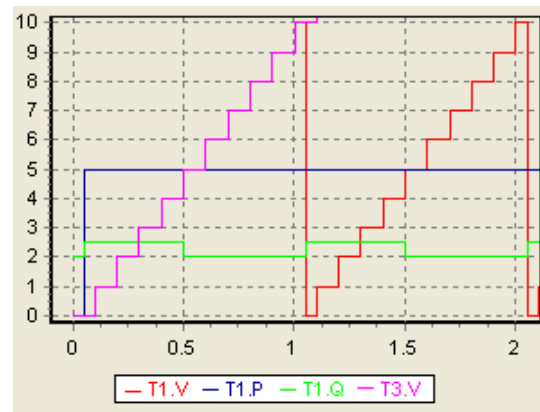
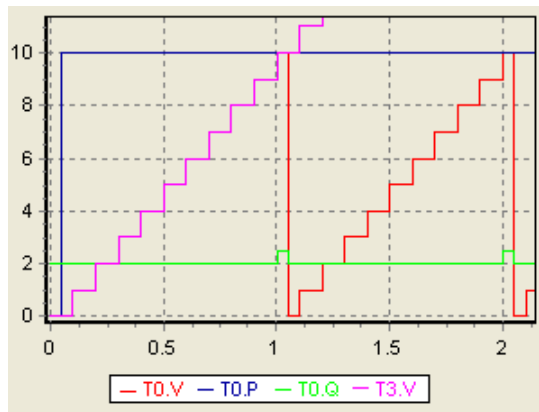
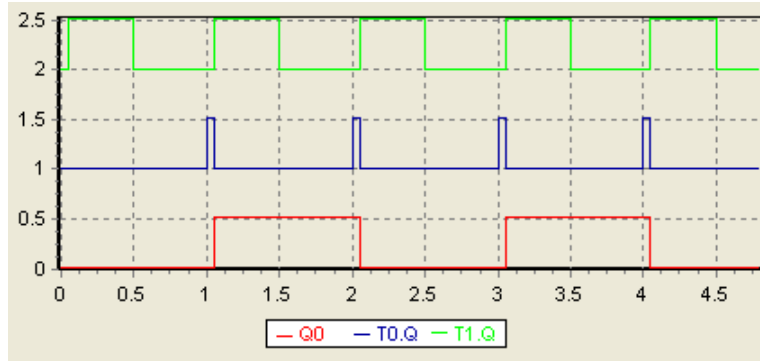
if sw0 = 0 then // inicializações

    t0.mode := TOn; // modo de atraso à activação
    t0.P := 10; // tempo de activação em décimos de segundo
    start t0; // inicio da contagem do tempo

    t1.mode := TOff; // modo de atraso à desactivação
    t1.P := 5;
    start t1;

end_if;

if t0.q then
    start t0;
    start t1;
    q0 := not q0;
end_if;
```



13.9. Máquina de Estados – Estados são bits com nome descritivo

```
(*****)
(*  StateMach2Bits.faproj - Maquina de Estados com 3 Estados  *)
(*****)
(*                                                                *)
(* Para testar este programa, comandar as entradas 1,2 e 3      *)
(* NormalRun é FALSE apenas no ciclo de inicialização          *)
(*                                                                *)
(* Os estados são os bits de memória: State1, State2 e State3   *)
(*                                                                *)
(* A transição de 1 para 2 ocorre quando In1=True              *)
(* A transição de 2 para 3 ocorre quando In2=True              *)
(* A transição de 3 para 1 ocorre quando In3=True              *)
(*                                                                *)
(* Out1 é activada quando State1=True                           *)
(* Out2 é activada quando State2=True                           *)
(* Out3 é activada quando State3=True                           *)
(* Out4 é activada em State2 e em State3                        *)
(*****)

// Inicializações
IF (NOT NormalRun) THEN
    NormalRun:=True;
    State1:=True;  // Estado inicial do StateOne
    State2:=False; // Estado inicial do StateTwo
    State3:=False; // Estado inicial do StateThree
END_IF;

// Evolução da máquina de estados
// Obs: No caso genérico, a Condição de Evolução
// entre estados pode ser uma condição complexa ou
// eventualmente um bit auxiliar de memória. o
// Neste exemplo, utiliza-se simplesmente uma entrada

IF State1 AND In1 THEN
    State1:=False;
    State2:=True;
ELSIF State2 AND In2 THEN
    State2:=False;
    State3:=True;
ELSIF State3 AND In3 THEN
    State3:=False;
    State1:=True;
END_IF;

// Atribuição de saídas da máquina de estados
Out1:=State1; // equação booleana (MAXTERMs, p. ex.)
               // dos estados em que A saída OutOne está activa
Out2:=State2;
Out3:=State3;
Out4:=State2 OR State3;
```

13.10. Máquina de estados – estado num inteiro

```
(*****)
(*  StateMach1Word.faproj - Maquina de Estados com 3 Estados  *)
(*****)
(*                                                                *)
(* Para testar este programa, comandar as entradas 1,2 e 3      *)
(* NormalRun é FALSE apenas no ciclo de inicialização          *)
(*                                                                *)
(* O inteiro de memória State contém o número do estado activo *)
(*                                                                *)
(* A transição de 1 para 2 ocorre quando In1=True              *)
(* A transição de 2 para 3 ocorre quando In2=True              *)
(* A transição de 3 para 1 ocorre quando In3=True              *)
(*                                                                *)
(* Out1 é activada quando State=0                               *)
(* Out2 é activada quando State=1                               *)
(* Out3 é activada em State=1 e em State=2                     *)
(*****)

IF (NOT NormalRun) OR (State<0) OR (State>2) THEN // inicializar
    NormalRun:=TRUE;
    State:=1; // exemplo: nascer no estado 1
END_IF;

IF State=0 AND InBit1 THEN
    State:=1;
ELSIF State=1 AND InBit2 THEN
    State:=2;
ELSIF State=2 AND InBit3 THEN
    State:=0;
END_IF;

Out1:=(State=0);
Out2:=(State=1);
Out3:=(State=1) OR (State=2);
```


13.11. Máquinas de estados com temporizadores

```
(*****)
(* StateMach3Timers.faproj - Máquinas de estados e temporizações *)
(*****)

// Inicializações
IF (NOT NormalRun) THEN
    NormalRun:=True;
    State1:=True;  // Estado inicial do State1
    State2:=False; // Estado inicial do State2
    State3:=False; // Estado inicial do State3
END_IF;

// Evolução da máquina de estados
IF State1 AND ((CurrentTime-Time1)>10) THEN
    State1:=False;
    State2:=True;
ELSIF state2 AND ((CurrentTime-Time2)>20) THEN
    State2:=False;
    State3:=True;
ELSIF State3 AND ((CurrentTime-Time3)>30) THEN
    State3:=False;
    State1:=True;
END_IF;

// Atribuição de saídas da máquina de estados
Out1:=State1;
Out2:=State2;
Out3:=State3;

// Lançar timers nos flancos dos estados
// Se o estado mudou neste ciclo, então...
IF (State1<>PrevState1) THEN
    Time1:=CurrentTime;
END_IF;
IF (State2<>PrevState2) THEN
    Time2:=CurrentTime;
END_IF;
IF (State3<>PrevState3) THEN
    Time3:=CurrentTime;
END_IF;

// Variáveis auxiliares
PrevState1:=State1;
PrevState2:=State2;
PrevState3:=State3;
```

13.12. Operações ao flanco ascendente e descendente

```
(*****  
(* RiseEdgeFallEdge.faproj - Utilização de RE e FE *)  
(* Para testar este programa, comandar entradas 0 e 1 *)  
(***)  
IF RE In0 THEN  
    Out0:=NOT Out0; // On rising edge of In0 => flip Out0  
END_IF;  
IF FE In0 THEN  
    Out1:=NOT Out1; // On falling edge of In0 => flip Out1  
END_IF;  
IF RE Out0 THEN  
    Out4:=NOT Out4; // On rising edge of Out0 => flip Out4  
END_IF;  
IF FE Out1 THEN  
    Out5:=NOT Out5; // On falling edge of Out1 => flip Out5  
END_IF;
```

13.13. Temporizadores do Utilizador

```
(*****  
(* UserTimer.faproj - Utilização de temporizadores *)  
(* Premir continuamente F8 e comandar a entrada 0 *)  
(***)  
  
print(CycleCount);print(' - ');println(TenHzSystemTimer);  
  
IF CycleCount=0 THEN  
    MyTimer.P:=40; // Preset to 4 seconds; assume timer is TOn  
END_IF;  
  
IF RE in0 THEN  
    START MyTimer;  
END_IF;  
  
print('My Timer.V='); print (MyTimer.V);print(chr(9));  
print('My Timer.P='); print (MyTimer.P);print(chr(9));  
print('My Timer.Q='); print (MyTimer.Q);println('');
```

13.14. Máquinas de estado e Temporizadores do Utilizador

```
(*****  
(* States+Timers.faproj - *)  
(* Máquinas de Estado misturadas com temporizadores *)  
(* Comuta periodicamente entre estados *)  
(***)  
IF (StateNumber=0) THEN // inicializações  
    StateNumber:=1;  
    START TimeState;  
    START Pulse;  
    TimeState.mode:=TOn; // desnecessário pois TOn é o Default  
    TimeState.P:=30; // décimas de segundo  
    Pulse.mode:=TOff;  
    Pulse.P:=5;  
END_IF;  
  
IF (StateNumber=1) AND TimeState.Q THEN  
    StateNumber:=2;  
    START TimeState;  
    START Pulse;  
ELSIF (StateNumber=2) AND TimeState.Q THEN  
    StateNumber:=1;  
    START TimeState;  
    START Pulse;  
END_IF;  
  
OutState1 := (StateNumber=1);  
Out1Pulse := (StateNumber=1) AND Pulse.Q;  
OutState2 := (StateNumber=2);  
Out2Pulse := (StateNumber=2) AND Pulse.Q;
```

13.15. Máquinas de estado e Temporizadores (exemplo melhorado)

```

(*****)
(* StateMach4Timers.faproj - Máquinas de estados e temporizações *)
(* State=0 => Init *)
(* State=1 => 1 seg ; State=2 => 2 seg ; State=3 => 3 seg *)
(* FEUPAutom 1.0 ou superior - Março de 2007 *)
(*****)

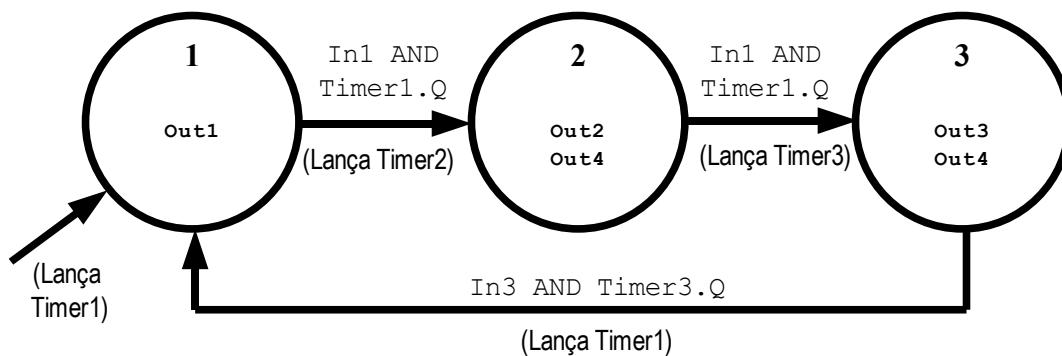
// Inicialização da máquina estados
IF (CurState=0) THEN
  CurState:=1;
  START Timer1;
  Timer1.mode:=TON;
  Timer2.mode:=TON;
  Timer3.mode:=TON;
  Timer1.P:=10;
  Timer2.P:=20;
  Timer3.P:=30;

// Evolução normal da máquina de estados
ELSIF (CurState=1) AND (Timer1.Q) AND In1 THEN
  CurState:=2;
  START Timer2;
ELSIF (CurState=2) AND (Timer2.Q) AND In2 THEN
  CurState:=3;
  START Timer3;
ELSIF (CurState=3) AND (Timer3.Q) AND In3 THEN
  CurState:=1;
  START Timer1;
END_IF;

// Atribuição de saídas da máquina de estados
Out0 := (CurState=0);
Out1 := (CurState=1);
Out2 := (CurState=2);
Out3 := (CurState=3);
Out4 := (CurState=2) OR (CurState=3);

OutTimer1:=Timer1.Q;
OutTimer2:=Timer2.Q;
OutTimer3:=Timer3.Q;

```



14. Dicas para programação em ST

- Não esquecer que as atribuições utilizam `:=` e que as comparações utilizam `=`
- A linguagem ST não é sensível a maiúsculas e minúsculas. No entanto, para ter 'syntax highlighting', utilize as primitivas da linguagem em Maiúsculas. Geralmente as variáveis contêm maiúsculas e minúsculas misturadas, p. ex.: `UmaVariável`
- Nas expressões booleanas é obrigatório por (parêntesis) entre todas as expressões elementares. Especial cuidado com IFs e NOTs, p. ex.: `IF ((X=Y) AND ((NOT Z)=W)) THEN`
- Geralmente não é necessário ter ciclos dentro de um Script. Crie uma estrutura de programa baseado no teste de casos (diversos IFs). Deixe que o ciclo global funcione.
- Por vezes é interessante atribuir a uma variável determinado valor, por exemplo atribuir 42 à variável 'contador'. Vá ao início do script e escreva a atribuição em causa seguida de `exit`, p. ex: `contador:=42;exit;` seleccione Run Once e depois comente a linha ou apague-a. Para inicializar o sistema, utilize Clear memory, Clear IOs, Clear System.
- Escreva código legível:
 - Utilize uma instrução por linha
 - Inicie o código na margem esquerda
 - Utilize indentação correcta: após cada IF (ou outra instrução de bloco), chegue o código 2 espaços para a direita, após cada `ENF_IF`, reponha o código 2 espaços antes (ver exemplo).
 - Utilize nomes descritivos adequados para cada variável
 - Utilize comentários adequados. Apesar da norma ST mencionar apenas (* comentários *), o FEUPAutom permite ainda // comentários.

15. Situações conhecidas e como as contornar

- O ficheiro do projecto '*.faproj' é um ficheiro de texto que pode ser aberto com o 'notepad'
- Por vezes é possível ter janelas fora do écran. Para corrigir, editar o ficheiro de projecto e alterar a janela com problemas para valores tal como os seguintes:

```
[FMain]
top=10
left=10
...
[FVariables]
top=10
left=10
...
[FIOLEds]
top=10
left=10
```
- Existe um problema com o windows que causa que o FEUPAutom funcione mal quando as fontes do sistema estão em tamanho maior que 100%. Para contornar este problema do windows, nas propriedades avançadas do desktop, propriedades gerais, escolher fontes pequenas (tamanho normal).

16. *Funcionalidade dos bits do sistema*

- %S0 – comuta a cada ciclo
- %SW0 – contador de ciclos executados
- %S16 – activo durante um ciclo a cada 10 ms
- %SW16 – contador do sistema em décimos de segundo

17. Referência de ST

Bit instructions

Description	Function
:=	Bit assignment
OR	Boolean OR
AND	Boolean AND
XOR	Exclusive Boolean OR
NOT	Inversion
RE	Rising edge
FE	Falling edge
SET	Set to 1
RESET	Reset to 0

Numerical comparisons on words, double words and floating points

Description	Function
<	Strictly less than
>	Strictly greater than
<=	Less than or equal to
>=	Greater than or equal to
=	Equal to
<>	Different from

Integer arithmetic on words and double words

Description	Function
+	Addition
-	Subtraction
*	Multiplication
/	Integer division
REM	Remainder of the integer division
SQRT	Integer square root
ABS	Absolute value
INC	Incrementation
DEC	Decrementation

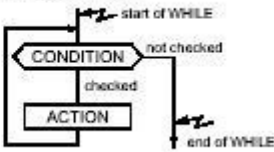
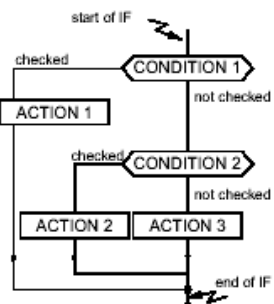
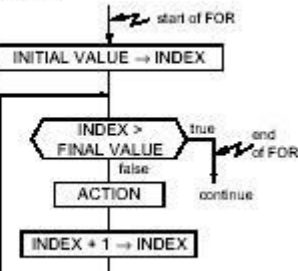
Arithmetic on floating points

Description	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
SQRT	Square root
ABS	Absolute value

Logic instructions on words and double words

Description	Function
AND	Logic AND
OR	Logic OR
XOR	Exclusive logic OR
NOT	Logic complement
SHL	Logic shift to left
SHR	Logic shift to right
ROL	Logic rotate to left
ROR	Logic rotate to right

Syntax	Operation
<pre>IF condition THEN actions ; END_IF;</pre>	<pre> graph TD Start([start of IF]) --> Condition{CONDITION} Condition -- checked --> Actions[ACTIONS] Condition -- not checked --> End([end of IF]) Actions --> End </pre>

<p>Syntax</p> <pre> WHILE condition DO action; END_WHILE; </pre>	<p>Mode of operation</p>  <pre> graph TD Start((start of WHILE)) --> Cond{CONDITION} Cond -- checked --> Act[ACTION] Act --> Cond Cond -- not checked --> End((end of WHILE)) </pre>
<p>Syntax</p> <pre> IF condition 1 THEN action1; ELSIF condition 2 THEN action2; ELSE action3; END_IF; </pre>	<p>Operation</p>  <pre> graph TD Start((start of IF)) --> Cond1{CONDITION 1} Cond1 -- checked --> Act1[ACTION 1] Act1 --> End((end of IF)) Cond1 -- not checked --> Cond2{CONDITION 2} Cond2 -- checked --> Act2[ACTION 2] Cond2 -- not checked --> Act3[ACTION 3] Act2 --> End Act3 --> End </pre>
<p>Syntax</p> <pre> FOR index := initial value TO final value DO action; END_FOR; </pre>	<p>Operation</p>  <pre> graph TD Start((start of FOR)) --> Init[INITIAL VALUE -> INDEX] Init --> Cond{INDEX > FINAL VALUE} Cond -- true --> End((end of FOR)) Cond -- false --> Act[ACTION] Act --> Inc[INDEX + 1 -> INDEX] Inc --> Cond </pre>

Operator priority rules

The table below gives the priority for evaluating a higher or lower priority expression.

Operator	Symbol	Priority
Parentheses	(expression)	Highest
Logic complement Inversion - on an operand + on an operand	NOT NOT - +	
Multiplication Division Modulo	* / REM	
Addition Subtraction	+ -	
Comparisons	<, >, <=, >=	
Comparison of equality Comparison of inequality	= <>	
Logic AND Boolean AND	AND AND	
Logic exclusive OR Boolean exclusive OR	XOR XOR	
Logic OR Boolean OR	OR OR	Lowest

Example :

NOT %MW3 * 25 AND %MW10 + %MW12

In this example, the NOT is performed on %MW3, then the result is multiplied by 25. The sum of %MW10 and %MW12 is calculated, then a logic AND is performed between the result of the multiplication and the addition.

Use of parentheses

Parentheses are used to modify the order in which operators are evaluated, for example to give an addition higher priority than a multiplication.

Example :

(%MW10 + %MW11) * %MW12

In this example, the addition will be performed before the multiplication.

Parentheses can be nested; there is no limit to the levels of nesting.

Parentheses can also be used to avoid incorrect interpretation of the program.

Example :

NOT %MW2 <> %MW4 + %MW6

By using operator priority rules, the following interpretation is obtained :

((NOT %MW2) <> (%MW4 + %MW6))

The user might well try to perform the following operation :

NOT (%MW2 <> (%MW4 + %MW6))

This example shows that parentheses can be used to clarify the program.

- fim do ficheiro de help do FEUPAutom -