ICAP - Interface de Comunicação para µAutómato Programável

Fernando J. F. C. de Sousa, Máximo A. S. de Oliveira, José Luis Azevedo, J. P. Estima de Oliveira

***Resumo*- Apresenta-se neste trabalho uma ferramenta de software, desenvolvida para computador pessoal, que substitui a consola de programação de um autómato programável.**

A ferramenta desenvolvida apresenta-se ao utilizador sobre uma interface gráfica em ambiente Windows.

***Abstract*- This work presents a software tool, developed for a personal computer, that can replace the programming console of a programmable logic controller (PLC).**

To the user, the tool appears as a graphic interface under the Windows environment.

I. INTRODUÇÃO

Hoje em dia as mais diversas tarefas tendem a ser controladas e/ou automatizadas, obrigando, em cada caso, ao estudo do processo que se pretende automatizar e ao desenvolvimento de um sistema de controlo adequado. Em muitas situações, os autómatos programáveis, mercê da sua versatilidade, expansibilidade e robustez, são os controladores de eleição.

Os autómatos programáveis são constituídos essencialmente por uma unidade de processamento, ligada a dispositivos de entrada/saída com isolamento galvânico, aos quais é possível ligar sensores, detectores, fins-de-curso, actuadores, etc, [1]. Nas configurações mais simples as entradas e as saídas são puramente digitais.

Um dispositivo ligado ao autómato, capaz de lhe fornecer um sinal, é um dispositivo de entrada. O ponto físico onde esse dispositivo é ligado ao autómato é genericamente designado por *entrada*. A cada entrada está associada uma posição e memória que contém o respectivo estado (ON ou OFF). Esta posição de memória é normalmente apelidada de *input bit.*

Na memória de um autómato existem também *output bits* que guardam os estados das saídas físicas, através das quais são actuados os dispositivos de saída.

Em qualquer modelo de autómato programável está implementado um conjunto de funções básicas. Por um lado existem funções booleanas como AND, OR e NOT, por outro encontram-se temporizadores, contadores e, por vezes, registos de deslocamento. Também é vulgar encontrarem-se intruções aritméticas, de manipulação de bits, bytes ou words, e instruções de controle do programa. Nas máquinas de gamas mais elevadas (que, muitas vezes, incluem entradas e saídas analógicas) encontra-se uma grande diversidade de outras instruções especiais.

Facilmente se conclui, assim, que um autómato pode substituir relés, temporizadores, contadores e outros dispositivos. O facto de ser controlado por "software" e, portanto, também programável garante-lhe uma grande versatilidade, mantendo no entanto a ideia do acesso discreto a cada um dos dispositivos de entrada/saída.

As linguagens de programação suportadas pelos autómatos dependem do modelo escolhido, podendo ir das simples listas de mnemónicas (idiossincráticas de família para família) e/ou diagramas de escada ("ladder logic") [2], até linguagens mais sofisticadas como os diagramas funcionais e o Grafcet [3], ou mesmo C. Hoje em dia, todas as linguagens suportadas pelos fabricantes tendem a obedecer, *grosso modo*, à norma IEC 1131-3 [4].

Para programar o autómato costuma utilizar-se uma consola que permite escrever e transferir programas para o autómato, monitorizar e alterar dados na memória. Nos modelos das gamas média e alta, os fabricantes disponibilizam igualmente ferramentas de programação utilizáveis em computadores possoais.

Quando este trabalho foi desenvolvido, os modelos SP-10, 16 e 20 da Omron eram apenas operáveis a partir de consola própria. É essa consola, que se pretende substituir com o ICAP - Interface de Comunicação para µAutómato Programável. Para isso, estabeleceram-se os seguintes objectivos:

- Caracterizar o protocolo de comunicação entre a consola e o autómato, já que este tipo de informação não é facultado pelo fabricante.
- Caracterizar o formato das diferentes instruções, tarefa necessária pelas mesmas razões do ponto anterior.
- Desenvolver uma ferramenta de "software", que proceda à validação sintáctica e lexical de um ficheiro contendo a sequência de instruções que formam um programa.
- Desenvolver uma ferramenta de "software", que codifique as diferentes instruções de um qualquer programa, em código inteligível pelo autómato.
- Desenvolver a ferramenta que, sobre um ambiente gráfico, permita a edição, validação, armazenamento, envio e recepção de programas para o autómato.
- Dotar a interface gráfica da possibilidade de monitorização e edição das várias zonas de memória.

II. A CONSOLA E O AUTÓMATO

A. Consola

A consola utilizada possui um teclado que permite aceder às instruções do autómato e a funções específicas de monitorização [5]. Tem ainda funções de "upload" e "download", podendo armazenar programas em cartões de memória de 16 Kbytes. Está dotada de um monitor de cristais líquidos onde são visualizáveis as instruções digitadas.

Com a consola é possível seleccionar dois modos de funcionamento: prgm e run. O primeiro permite edição, "upload" e "download" de programas. O segundo faz correr o programa existente na memória do autómato.

B. Autómato

O autómato utilizado foi um SP20 da OMRON [5]. Possui 12 entradas e 8 saídas. Tem um conjunto de 38 instruções e contém memória que permite construir programas até 240 instruções. Está dotado de 2 *analog timers*, 1 *reversible drum counter*, 16 *timers/counters*, 1 *shift register* de 16 bits e as instruções lógicas são igualmente de 16 bits. É programado por intermédio da consola descrita anteriormente.

C. Memória do Autómato

A memória do autómato está distribuída por sete áreas distintas:

12	*input bits* para receber sinais externos.
8	*output bits* para enviar sinais para o exterior.
204	*work bits* que, num programa, podem guardar os diferentes estados.
69	*bits* dedicados a funções específicas.
256	*data retention bits* onde pode ser guardada informação permanentemente (esta zona é alimentada por uma pilha).
128	*link relay*, zona usada para transferências de dados com outros autómatos.
16	*timers/counters* para definir temporizadores e contadores.

III. COMUNICAÇÃO

Dada a ausência de informacão, foi necessário determinar o protocolo usado na comunicação consola-autómato. Concluiu-se que a comunicação se processava nos seguintes termos:

9600 baud - 8 data bits, 1 start, 1 stop, even parity.

Determinou-se também o formato da trama de comunicação:

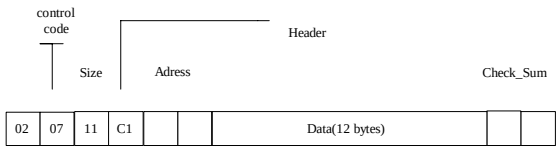


Fig. 1 - Trama de comunicação.

Estrutura da trama

02 início de trama.

Control code:

0F comunicação da consola para o autómato.

07 comunicação do autómato para a consola.

Size: comprimento da trama.

Header:

C1 a informação enviada pela consola destina-se a ser gravada na memória do autómato.

C3 a informação enviada não se destina a ser gravada na memória do autómato.

Address: Endereço destino da informação da trama.

Data: 12 bytes para transporte de informação.

Checksum: checksum da trama.

IV. ROTINAS DE BASE DO PROGRAMA ICAP

O programa ICAP foi desenvolvido de modo a possuir dois níveis distintos: um ao nível do utilizador, que consiste numa interface gráfica gerada em Visual Basic, e outro de mais baixo nível, que consiste em rotinas de comunicação, monitorização, interpretação e geração de código (Fig.3). Estas rotinas de base, que serão apresentadas de seguida, foram desenvolvidas em C++, recorrendo-se ainda ao uso das ferramentas *Lex* e *Yacc* para a construção de um tradutor.

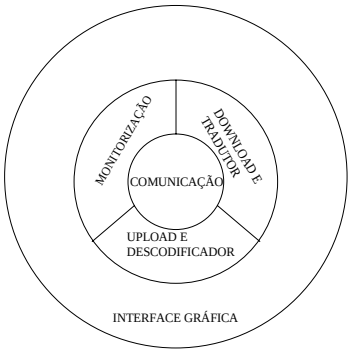


Fig. 2 - Diagrama da estrutura do programa ICAP.

A. Tradutor

Pretende-se transformar um conjunto de mnemónicas organizadas num ficheiro (programa), em código legível pelo autómato. Para tal recorreu-se a dois utilitários destinados a escrever compiladores - *Lex* e *Yacc* [6].

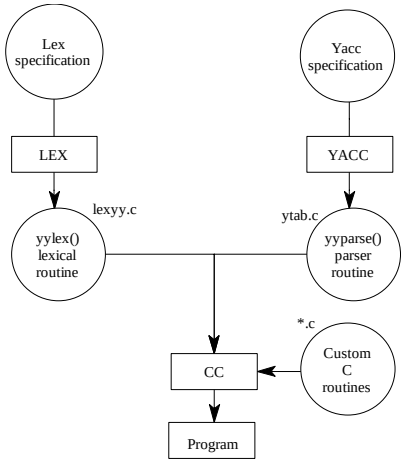


Fig. 3- Construção do tradutor.

Estes, como se ilustra na figura 3, aceitam um ficheiro de especificações e geram, respectivamente, uma rotina lexical-yylex() e uma rotina *parser*-yyparse(), que, quando compiladas com código construído pelo programador, constroem o tradutor desejado.

B. Descodificador

O descodificador lê a informação contida num ficheiro, tenta identificá-la com o código de instruções conhecidas e, caso o faça com sucesso, escreve-a noutro ficheiro.

Quando o código lido não é identificável com uma instrução (situação pouco provável, já que na recepção é verificado o *checksum*), a rotina pára a sua execução e devolve uma mensagem de erro.

Para proceder à interpretação do código armazenado no ficheiro, este é inicialmente carregado na memória do PC e é inicializado um ponteiro (sinalizador de início de instrução) para essa área da memória.

O método desenvolvido para identificar uma instrução (sinalizada pelo ponteiro), consiste numa comparação exaustiva da memória com o formato das várias instruções conhecidas. Como o código das diferentes instruções varia em tamanho e tem *bytes* variáveis, dependendo do tipo de operandos e do respectivo *range*, foi desenvolvida uma função especial para efectuar essa comparação.

Especificações da função:

- número de parâmetros de entrada variável.
- o primeiro parâmetro é uma *string* cujo formato é discriminatório dos parâmetros seguintes.
- formato do primeiro parâmetro:

‘*’- qualquer byte (não necessita de mais informação)

‘d’- um byte (discriminado nos parâmetros da função)

‘[]’- um byte descrito bit a bit, podendo estes serem variáveis ‘x’ - não necessita de informação complementar (Ex: [01x00xx1]).

Exemplo elucidativo da acção desta função

```
inf. na memória 4F   01   FF   49   00
                ↑
                ponteiro

func_comparação(“*dd[0x001001]*”,0x01,0xff);
```

O primeiro byte (endereçado na memória pelo ponteiro) pode ser qualquer um, já que está especificado com um ‘*’ (não necessita de mais informação); o segundo (ponteiro+1), como está especificado com um ‘d’, é obrigatoriamente o valor contido no segundo argumento da função(0x01); o mesmo acontece com o terceiro (pos+3) (0xff); o quarto terá o formato especificado entre []; finalmente, o quinto obedece ao procedimento do primeiro.

Se o conteúdo da posição de memória seguinte ao ponteiro, obedecer ao formato descrito no primeiro argumento da função e retantes argumentos, se existirem, esta função retorna 1, se não devolve 0.

A comparação com as diferentes instruções é executada com o seguinte procedimento:

```
Se(= A) Gera instrução A;
Se não Se(= B) Gera instrução B;
Se não Se(= C) Gera instrução C;
        .....
```

sendo: A,B,C,... os formatos respectivos das várias instruções.

‘Gera instrução *x*’ função que escreve em ASCII a instrução detectada, e que actualiza o ponteiro (sinalizando o início de nova instrução).

O procedimento descrito é executado até ser encontrada a instrução ‘END’(Fig.4).

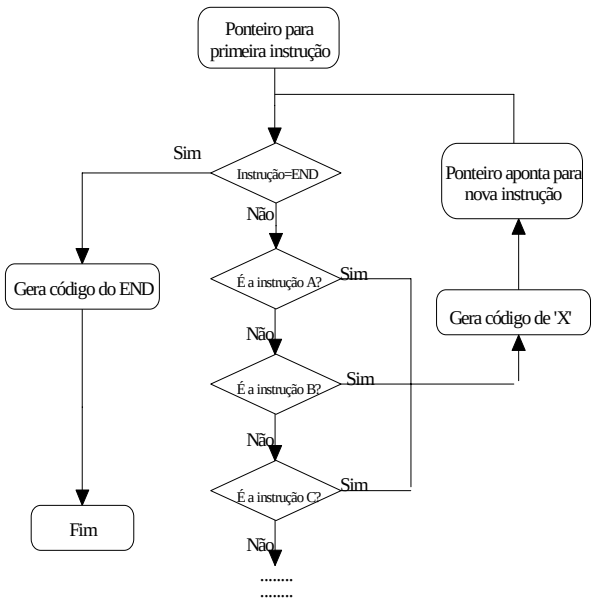


Fig. 4 - Descodificação das instruções.

A rotina ‘Geração de instrução’ é chamada com um identificador da instrução detectada na memória. Através

deste, selecciona um bloco que vai ser o responsável pela transformação para ASCII da instrução e pela actualização do ponteiro para a instrução seguinte. A transformação resultante (ASCII) é escrita num ficheiro. No final, quando a instrução 'END' é detectada, o ficheiro contém o texto de todo o programa.

C. Download

O envio de um programa para o autómato é feito através de 61 tramas: 58 tramas de dados, correspondentes a toda a memória do autómato desde o endereço 0xFB00, até 0xFDB8 (com o incremento de 12 por trama) e 3 tramas finais de controlo.

Inicialmente é construída uma estrutura que vai conter as 61 tramas atrás descritas (Fig.5).

De seguida, as tramas de dados são preenchidas com o código gerado pelo compilador, tendo cada uma capacidade para 12 bytes. O processo repete-se até ser encontrada uma instrução de *end*.

No final, na terceira trama de controlo, é escrito o *checksum* (2 bytes) de todos os dados transportados pelas tramas de *data*.

Resta agora o envio de cada uma das tramas para o autómato. Começa-se por inicializar a porta série para uma comunicação com *baudrate* de 9600, 1 bit de paridade (par), 1 *start bit* e 1 *stop bit*.

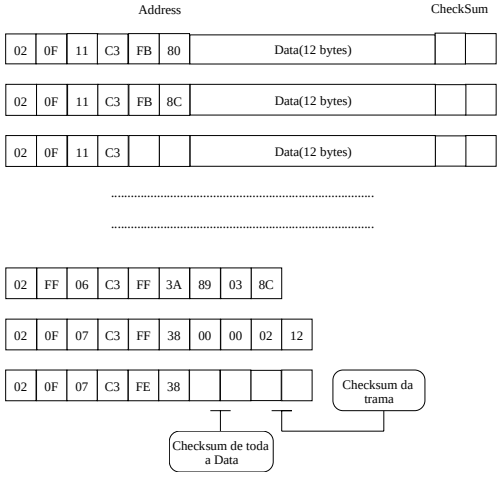


Fig. 5 - Estrutura de tramas do programa do autómato.

Em seguida, são enviadas as tramas previamente preenchidas com os dados do programa, intercalando entre cada uma delas uma trama de sincronismo:

02	A0	02	00	a4

Entre cada byte enviado, é imposto um *delay* de 10ms. Este valor foi encontrado experimentalmente, como sendo óptimo para o autómato receber o programa sem erros.

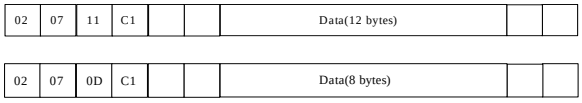
Após o envio de cada trama, aguarda-se a resposta do autómato durante 500 ms. Esgotado este período de tempo é de novo enviada a trama, repetindo-se este procedimento até à sexta tentativa. Falhadas todas as tentativas, é devolvida uma mensagem de erro e a execução do programa termina.

O fluxograma da Fig.6 exemplifica este procedimento.

D. Upload

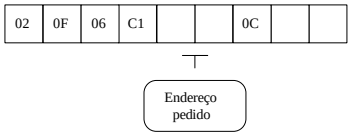
Estando disponíveis as ferramentas de edição, compilação e transferência de um programa para o autómato, surge a necessidade de receber no PC um programa vindo do autómato. Nesta secção vamos mostrar o procedimento para ler um programa contido na memória do autómato.

A recepção de um programa no PC, é feita através de 58 tramas de dados.



... ...

Para obter uma determinada trama é feito o pedido correspondente.



Ao envio de uma trama de pedido, o autómato responde com a trama de *data* correspondente ao endereço pedido.

Assim, fazendo pedidos desde o endereço 0xFB00

02	0F	06	C1	FB	80	0C	02	5F

até ao endereço 0xFE2C

02	0F	06	C1	FE	2C	0C	02	0E

e com incremento de endereço 12, obtemos toda a área de memória de programa.

Tal como no caso da transmissão, também aqui é devolvida uma mensagem de erro, se se esgotarem as 6 tentativas de recepção de uma trama (Fig.7).

A cada trama recebida, é extraída a informação e escrita num ficheiro. Terminada com sucesso a execução do programa de recepção, obtém-se um ficheiro que contém o código do programa residente na memória do autómato.

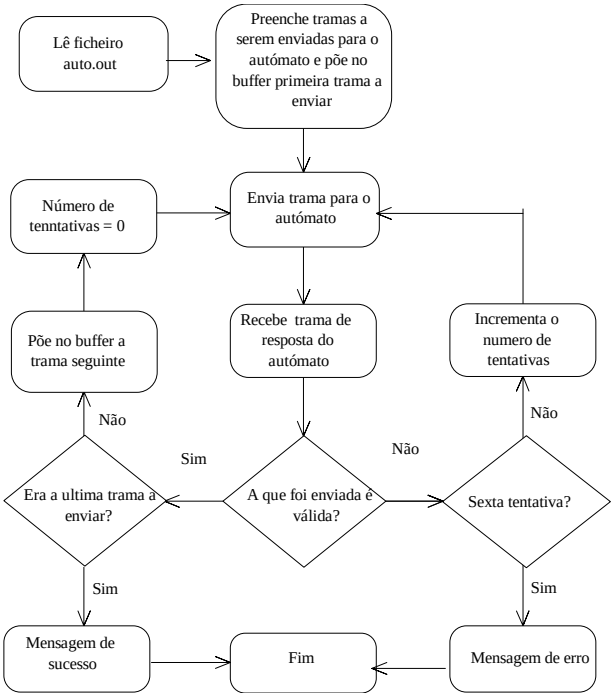


Fig. 6 - Envio de um programa para o autómato.

V. MONITORIZAÇÃO

A monitorização do autómato tem como objectivo conhecer o seu estado interno como, por exemplo, saber quais os *timers/counters* usados, qual o conteúdo de certas zonas da memória, ou qual o estado das entradas e das saídas.

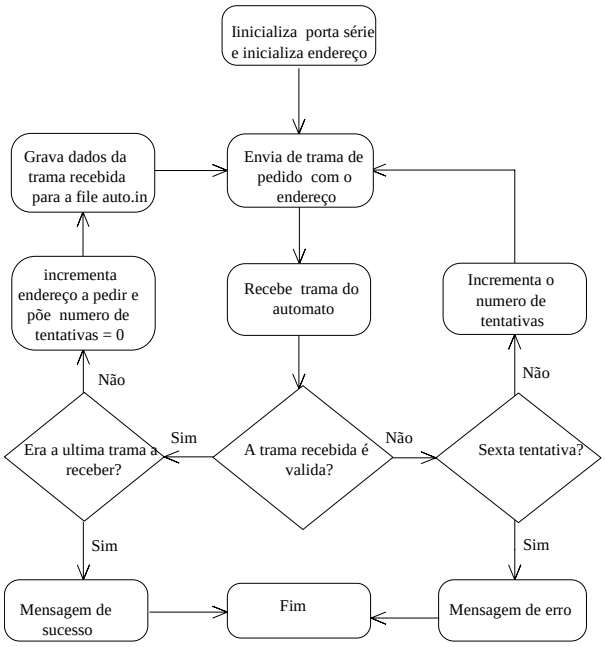
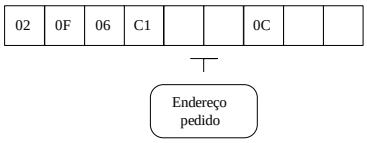


Fig. 7 - Recepção de um programa do autómato.

Toda esta informação pode ser obtida enviando para o autómato "tramas-pedido" adequadas. Essas tramas indicam o endereço da memória do autómato sobre a qual se pretende obter informação.

Por exemplo, para se receber o estado da área de memória DR são enviadas para o autómato tramas cujo endereço varia desde 0xFF40 até 0xFF58, com incremento de 8:



Por cada trama pedido enviada, o autómato devolve o conteúdo do endereço correspondente, em tramas do tipo:

VI. INTERFACE GRÁFICA

O ambiente gráfico foi desenvolvido com base no Microsoft Visual basic 3.0, com o intuito de fornecer ao utilizador um *feedeback* fiável e preciso sobre a tarefa em execução.

Possui funcionalidades de edição, monitorização e invocação de tarefas necessárias ao sistema.

Num primeiro contacto, a interface apresenta-se ao utilizador conforme a Fig.8, onde se destaca o editor que possibilita a edição e visualização de programas, possuindo facilidades de *copy & paste*, informação sobre o número da linha em edição e nome do ficheiro em uso.

Está disponível uma barra de botões associada ao editor, possibilitando a edição rápida de várias funcionalidades. A identificação das mesmas surge momentaneamente no ecrã, quando o rato passa por cima do botão respectivo. Quando um dos botões é premido surge, no editor, o texto correspondente à instrução.

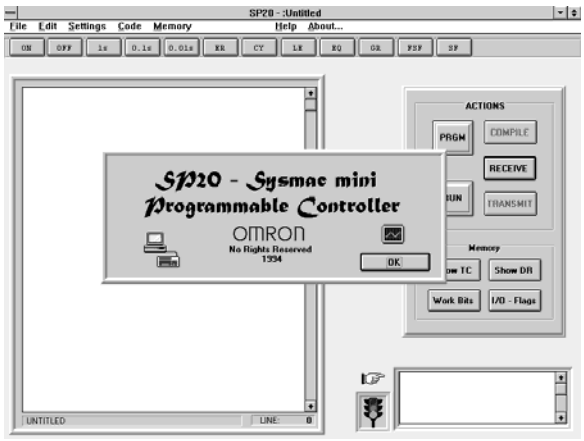


Fig. 8 - Aspecto da interface de entrada

A. Monitorização

A monitorização do estado interno do autómato é feito através de janelas, as quias permitem visualizar o conteúdo de áreas específicas da memória do autómato.

A invocação das janelas respectivas é feita, como atrás se descreveu, usando o painel de comandos, os menus ou combinações de teclas.

B. Temporizadores/Contadores (TC)

A janela apresentada na figura 9 permite ao utilizador monitorizar o estado de todos os TC's (*timers* e *counters*). O botão LOAD executa uma rotina que lê do autómato a informação necessária e apresenta-a ao utilizador numa forma clara.

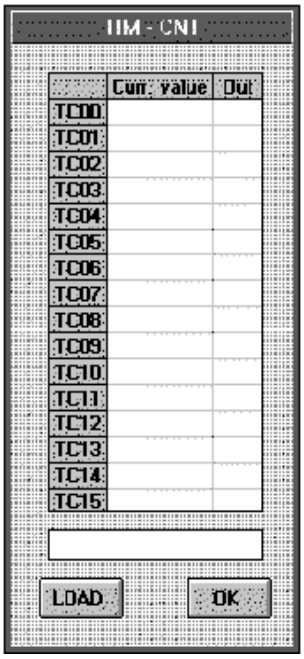


Fig. 9 - Monitorização dos *timers/counters*

Esta janela mostra uma grelha onde são assinalados, através de um símbolo (neste caso, uma lâmpada) na coluna *out*, todos os TC´s que estão a ser usados pelo programa a correr no autómato. Se um dado TC está, num dado momento, activo, a lâmpada apresenta a cor amarela, e vermelha no caso contrário. A coluna *Curr. value* apresenta o valor actual de cada TC.

C. Memória DR

A importância desta área de memória foi já referida. Além das facilidades de monitorização foram também criadas facilidades de edição, com as opções de envio para o autómato ou de gravação em ficheiro.

A janela, apresentada na figura 10, ilustra uma grelha que permite a visualização dos 256 bits da área DR, organizada em palavras de 16 bits. A grelha possui ainda uma coluna onde é escrito, em hexadecimal, o valor de cada palavra.

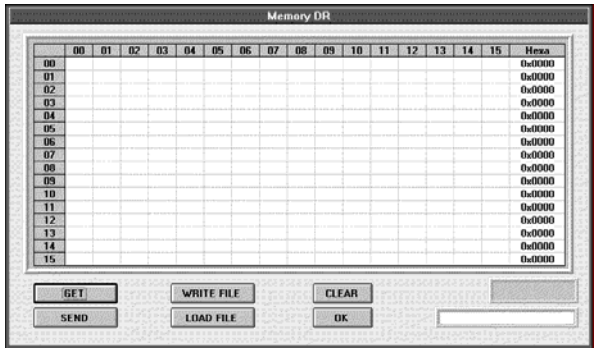


Fig. 10 - Monitorização/Edição da área de memória DR.

A edição é feita directamente sobre a grelha com um *double click* do rato, o que provoca o *toggle* do conteúdo da célula seleccionada. A coordenada do bit, correspondente a essa célula é mostrada junto ao canto inferior direito da grelha. Os botões existentes são:

Este botão acciona a rotina que lê e mostra a área de memória DR do autómato.

Este botão invoca o programa que transfere o conteúdo da grelha para a memória do autómato.

Permite gravar num ficheiro a informação apresentada na grelha.

Lê um ficheiro pré-gravado com um mapa de memória DR e apresenta-o na grelha.

Coloca a zero todas as células da grelha.

Abandona a janela.

D. Workbits

Nesta janela (Fig.11) são apresentadas grelhas que permitem, através do botão LOAD, visualizar o conteúdo dos diferentes *workbits*.

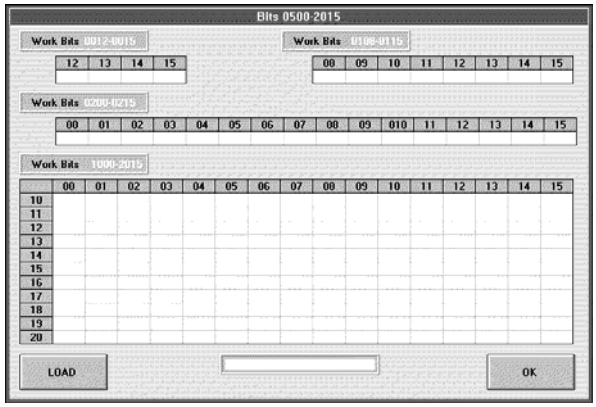


Fig. 11 - Monitorização dos *Workbits*.

E. I/O + Flags

As entradas, as saídas e as *flags* do autómato são monitorizadas nesta janela. Quando o utilizador acciona o botão LOAD, são lidos do autómato os estados referentes a cada uma delas.

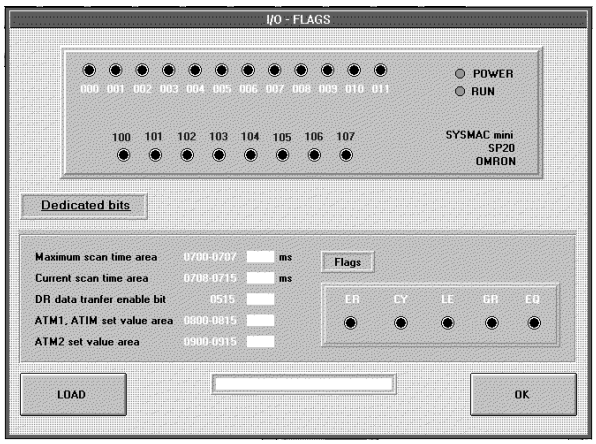


Fig. 12 - Monitorização de I/O e das *flags*.

Nesta janela é representado simbolicamente o autómato com as suas entradas, saídas e *flags*. Quando qualquer uma delas está ON é visualizada como se de um *led* vermelho (aceso) se tratasse.

Como se pode verificar, esta janela também apresenta algumas outras informações úteis.

REFERÊNCIAS

[1] Michel, Gilles, "Programmable Logic Controllers - Arquitecture and Applications", John Wiley & Sons, Chichester, 1990.

[2] Warnock, Ian G., "Programmable Controllers - Operation and Application", Prentice Hall, Hertfordshire, 1988.

[3] David, R. and H.Alla, "PetriNets and Grafcet: Tools for Modelling Discrete Event Systems", Prentice Hall, London, 1992.

[4] IEC 1132-3, International Standards, Programmable Controllers - part 3: programming languages, IEC, 1993.

[5] Omron, SP10/SP16/SP20 Operation Manual, Omron Corporation, Tokyo, 1992.

[6] Mason, Tony and Doug Brown, "Lex & Yacc", O'Reilly & Associates Inc., Sebastopol, 1990.