IMPLEMENTAÇÃO DE SUPERVISORES EM MICROCONTROLADORES: UMA ABORDAGEM BASEADA NA TEORIA DE CONTROLE DE SISTEMAS A EVENTOS DISCRETOS

CARLOS A. TEIXEIRA[(1)], ANDRÉ B. LEAL[(2)], ANTONIO H. DE SOUSA[(2)]

[(1)]Setor de Pesquisa & Desenvolvimento - Embraco Electronic Controls - EECON
Dona Francisca, 8300, Módulo 1 / Bloco B, 89239-270, Joinville, SC, Brasil

[(2)]Departamento de Engenharia Elétrica, Universidade do Estado de Santa Catarina - UDESC
Campus Universitário Prof. Avelino Marcante s/n, Bairro Bom Retiro, 89223-100, Joinville, SC, Brasil

E-mails: carlos_a_teixeira@embraco.com.br, {leal, heron}@joinville.udesc.br

Abstract — This paper proposes a framework and a procedure for the implementation of supervisors on microcontrollers. The methodology uses data search in tables located in program memory and aims to reduce the hardware requirements. The supervisors are designed based on the theory of supervisory control of Discrete-Event Systems, making possible to divide complex problems in a set of smaller problems. The application of the technique is illustrated using a problem called Digital Auto-Reverse Tape-Deck.

Keywords — Discrete-event systems, supervisory control, embedded systems, assembly language, automatic code generation.

Resumo — O artigo propõe uma estrutura e um procedimento para a implementação de supervisores em microcontroladores. A metodologia utiliza busca de dados em tabelas armazenadas em memória de programa e visa reduzir os requisitos de *hardware*. Os supervisores são projetados com base na teoria de controle supervisório de Sistemas a Eventos Discretos, o que possibilita dividir problemas complexos num conjunto de problemas menores. A aplicação da técnica é ilustrada utilizando um problema intitulado Toca-Fitas Auto-Reverso Digital.

Palavras-chave — Sistemas a eventos discretos, controle supervisório, sistemas embarcados, linguagem *assembly*, geração automática de código.

1 Introdução

Dentre os principais benefícios do uso de técnicas de modelagem no desenvolvimento de *software*, pode-se citar a organização do raciocínio em função da existência de um processo visual, maior depuração de erros na fase de concepção, facilidade de compreensão por outros profissionais, existência de ferramentas de verificação e análise de desempenho e possibilidade de geração automática de código.

A aplicação da teoria de Sistemas a Eventos Discretos (SEDs) (Wonham, 2005) na modelagem e controle de sistemas embarcados, proporciona vantagens adicionais às acima mencionadas, por disponibilizar ferramentas de *software* para cálculo automático de supervisores a partir de modelos pequenos da planta e das especificações de controle, tornando possível o tratamento de problemas complexos como um conjunto de problemas menores.

O objetivo do presente trabalho é propor uma estrutura e um procedimento para implementação de supervisores em microcontroladores de baixo custo (< 0,5 US$), com memória reduzida (≤ 2 kbytes) e pequena taxa de processamento (≤ 1 MIPS). Microcontroladores com essas características são bastante utilizados no controle eletrônico de eletrodomésticos. A implementação é realizada utilizando *software* em código *assembly*.

Emprega-se a teoria de controle supervisório de SEDs de forma a projetar um supervisor que garanta o cumprimento das especificações de controle. Na implementação do supervisor propõe-se o uso de tabelas em memória de programa e rotinas de análise dos dados contidos nessas tabelas.

São referências nesse tipo de iniciativa os trabalhos realizados por Wood (2005) e Soares (1999). Wood propõe a metodologia *Initiated-Event* para implementação de supervisores em microcontroladores. Essa metodologia é baseada na teoria de SEDs, mas no entanto difere do presente trabalho ao utilizar uma forma particular de definição do espaço de eventos, na qual todos os eventos são considerados controláveis. Wood aplica tal técnica no exemplo *Abstract Vending Machine*, entre outros. Soares propõe uma metodologia de implementação de estadogramas em sistemas embarcados, tratando o exemplo Controlador de um Portão de Segurança. Ambas as propostas visam redução de código e permitem a geração automática de *software* em código *assembly*.

O restante do artigo está organizado como segue. A próxima seção fornece um breve resumo da teoria de modelagem e controle supervisório de SEDs. A seção 3 apresenta a metodologia para implementação de supervisores. A seção 4 trata um exemplo de aplicação dessa metodologia. Por fim, apresentam-se as conclusões.

2 Modelagem e Controle Supervisório de SEDs

O comportamento de um SED pode ser modelado por um autômato, uma quíntupla $G = (Q, \Sigma, \delta, q_0, Q_m)$, onde Q é um conjunto finito de estados q_i, Σ é um conjunto de eventos σ_j, q_0 é um estado inicial, Q_m é um conjunto de estados marcados ($Q_m \subseteq Q$) e $\delta = Q$

$\times \Sigma \rightarrow Q$ é uma função de transição, possivelmente parcial, por ser não necessariamente definida para todo evento σ_j em cada estado q_i. Estados marcados representam tarefas completas, de modo que o alcance de um estado marcado significa a finalização de uma determinada tarefa. A Figura 1 apresenta a representação visual de um autômato.

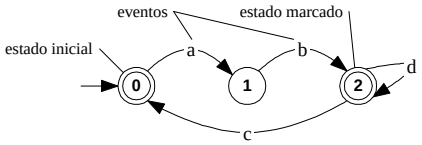

Figura 1. Representação de SEDs Através de Autômatos.

A sincronização da ocorrência de eventos em 2 ou mais autômatos é feita por meio do operador composição síncrona, denotado por // (Wonham, 2005). Os eventos comuns a Σ_1 e Σ_2 só podem ser executados sincronamente nos autômatos G_1 e G_2. Os demais eventos ocorrem assincronamente (de modo independente) em cada autômato.

A modelagem de um SED considera o sistema como sendo uma planta G que pode ser composta de diversas sub-plantas G_i, onde $G = G_1 // G_2 // ... // G_n$.

O controle da planta G tem como base a definição de um conjunto de especificações que ditam o comportamento desejado para a planta. Tal conjunto de especificações pode ser composto de diversas sub-especificações E_i, de modo que a especificação global é dada por $E = E_1 // E_2 // ... // E_n$. A linguagem alvo K, definida por $G // E$, representa as especificações de controle que são contidas na linguagem da planta G.

A teoria de controle supervisório de SEDs visa sintetizar um supervisor ótimo (minimamente restritivo e não bloqueante), que garanta que a planta sob supervisão se comporte de acordo com as especificações de controle (Cassandras e Lafortune, 1999).

Um supervisor minimamente restritivo limita o comportamento da planta à somente comportamentos desejados (definidos pela especificação global E). Um supervisor não bloqueante se caracteriza por nunca alcançar um estado no qual nenhuma tarefa possa ser finalizada.

No presente trabalho, as operações de produto síncrono e cálculo do supervisor foram feitas usando o *software* GRAIL (Raymond e Wood, 1995), que consiste numa biblioteca de funções em C++ para manejo de autômatos, expressões regulares e linguagens finitas. Tal *software* representa um SED por um conjunto de *strings* num arquivo texto. Por exemplo, o autômato da Figura 1 no GRAIL é representado pelas *strings* da Figura 2. A simbologia *(START) |- 0* indica que o estado *0* é o estado de partida. A simbologia *0 -| (FINAL)* e *2 -| (FINAL)* indica que os estados *0* e *2* são marcados. Cada linha do tipo $q_d \ \sigma \ q_a$ corresponde à uma transição do autômato (a ocorrência do evento σ no estado de partida q_d leva ao estado de chegada q_a).

```
(START) |- 0
0   a   1
1   b   2
2   d   2
2   c   0
0 -| (FINAL)
2 -| (FINAL)
```

t_1, t_2, t_3, t_4

t_n = transição n

Figura 2. Representação do Autômato da Figura 1 no GRAIL.

3 Implementação de Supervisores

A presente proposta de implementação de supervisores baseia-se na estrutura apresentada na Figura 3.

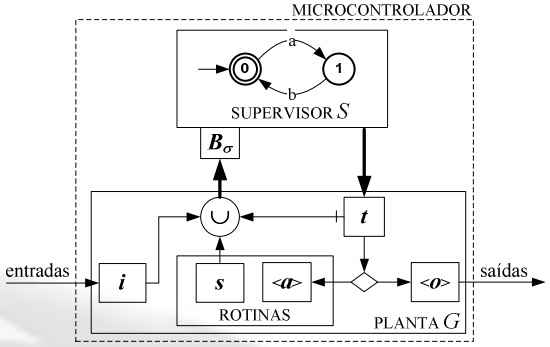

Figura 3. Proposta de Estrutura de Implementação.

Esta estrutura contém um supervisor S que trata eventos do tipo entrada *i*, sinalização *s* e tarefa *t*. Este supervisor S pode habilitar ou desabilitar eventos tarefa *t*. A ocorrência de um evento tarefa requisita a escrita numa porta de saída por meio de um *flag* de saída <*o*> ou requisita a realização de uma determinada ação dentro de uma rotina por meio de um *flag* de ação <*a*>.

Tal estrutura insere-se num nível de implementação onde outras rotinas de *software* com finalidades diversas também estão inseridas. Considera-se que interrupções sinalizadas por periféricos do microcontrolador encontram-se num nível inferior de implementação, não sendo assim modeladas na estrutura em questão.

Um evento de entrada *i* é disparado pela ocorrência de uma transição num pino de entrada digital. Essa transição pode ocorrer, por exemplo, ao ser pressionada uma tecla numa interface com o usuário.

Um evento de sinalização *s* é disparado dentro de uma rotina de *software* e visa informar o supervisor sobre a ocorrência de uma situação relevante. Um evento de sinalização *s* é disparado, por exemplo, quando um contador alcança seu valor máximo de contagem.

A habilitação de um evento tarefa *t* permite a ocorrência desse evento na planta G. Ao ocorrer, esse evento tarefa é colocado no *buffer* B_σ, de forma que o supervisor possa ser atualizado.

Um *flag* de saída <*o*> produz uma mudança de estado num pino de saída digital, atuando numa carga ligada ao sistema, por exemplo, um motor elétrico.

Um *flag* de ação <*a*> requisita à uma rotina de *software* a realização de uma determinada ação. Um *flag* de ação <*a*> pode, por exemplo, requisitar a reinicialização ou pausa de uma contagem de tempo.

Cada evento é associado com um rótulo hexadecimal. Por exemplo, a ocorrência de uma transição num determinado pino de entrada digital, dispara um evento de entrada *i* com um rótulo n_{hex}. A ocorrência de outra transição nesse mesmo pino dispara um novo evento de entrada *i* com mesmo rótulo n_{hex} e somente transições nesse pino específico geram eventos de entrada *i* com tal rótulo n_{hex}.

A definição se um evento tarefa *t* atua em um *flag* de saída <*o*> ou em um *flag* de ação <*a*> é feita por meio da análise de rotulação. Uma possibilidade é utilizar a convenção de que eventos tarefa com rótulo ≤ m_{hex} atuam em *flags* de saída <*o*> e eventos tarefa com rótulo > m_{hex} atuam em *flags* de ação <*a*>.

A Figura 4 apresenta uma proposta de diagrama de estados do nível de implementação.

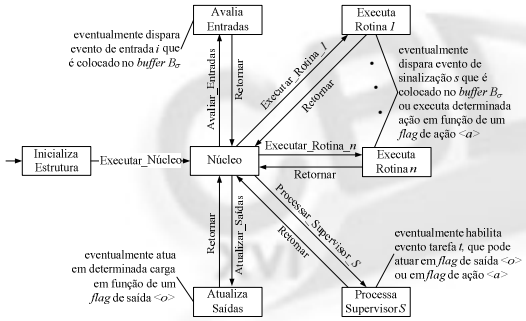

Figura 4. Proposta de Diagrama de Estados.

Com o objetivo de não perder informações por atrasos de processamento, utiliza-se um *buffer* B_σ para armazenar os eventos σ que aguardam ser processados pelo supervisor. Tal *buffer* deve ter um número de posições proporcional à velocidade do processador e ao maior ciclo de chamada e execução da rotina do supervisor. O supervisor é processado para cada evento σ que chega ao *buffer* B_σ, até que este *buffer* fique vazio. Durante o processamento dos eventos presentes no *buffer*, eventos não controláveis possuem tratamento prioritário.

Ao processar um evento σ o supervisor *S* transita de um estado de partida q_d para um estado de chegada q_a. Se o estado de chegada for um estado não marcado q_{nm}, ao transitar de estado o supervisor habilita um evento tarefa *t*, que ocorre na planta *G* e faz com que o processamento continue, até uma situação na qual *S* alcança um estado marcado q_m, considerado uma tarefa completa do sistema.

Na implementação realizada, a leitura de eventos de entrada foi desabilitada durante o processamento do supervisor. Outra alternativa de implementação consiste no uso de um *buffer* para armazenar eventos de entrada, que são lidos somente quando se alcança um estado marcado.

O supervisor *S* é implementado por meio de uma tabela *T* alocada em memória de programa, criada com base no conjunto de *strings* que descreve o supervisor. Tal tabela possui grupos de dados que contêm o estado de partida q_d, o evento σ e o estado de chegada q_a, para cada transição do supervisor. A tabela *T* possui também, implicitamente, a informação do estado inicial q_0 (sempre rotulado como estado *0*), quais estados são marcados (estados pares se o estado inicial for marcado) e quais são não marcados (estados ímpares se o estado inicial for marcado). A Figura 5 apresenta a tabela *T* correspondente às *strings* da Figura 2. O tamanho da tabela *T* é sempre 3 vezes o número de transições do supervisor *S*.

Ponteiro	*Dado*		
0	*0*	q_d	t_1
1	*a*	σ	
2	*1*	q_a	
3	*1*		t_2
4	*b*		
5	*2*		
6	*2*		t_3
7	*d*		
8	*2*		
9	*2*		t_4
10	*c*		
11	*0*		t_n = transição n

Figura 5. Tabela *T* das *Strings* da Figura 2.

O processamento do supervisor é feito acessando a tabela *T*, conforme fluxograma da Figura 6.

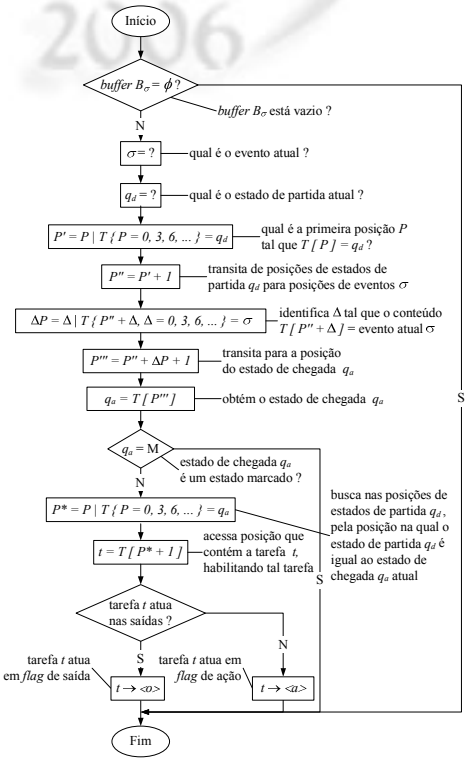

Figura 6. Fluxograma de Processamento do Supervisor *S*.

A notação $\alpha = T[\beta]$ significa que α corresponde ao dado contido na posição β da tabela *T*. A notação

$P \mid T\{P = i, j, k, ...\} = \gamma$ significa que P corresponde à primeira posição da tabela T, dentre as possíveis indicadas ($i, j, k, ...$), na qual a igualdade é satisfeita, ou seja, na qual o conteúdo é o dado γ.

4 Aplicação da Metodologia

A aplicação da metodologia proposta é ilustrada através de um exemplo de sistema embarcado intitulado Toca-Fitas Auto-Reverso Digital, que possui 17 eventos (*st, pl, rec, rew, rev, acg, rcg, acr, rcr, hg, dg, gh, grh, grah, gah, rb e p*). Os eventos *st, pl, rec, rew, rev* (relacionados às teclas), os eventos *hg* e *dg* (relacionados ao sensor de proteção contra gravação) e o evento *rb* (relacionado ao motor) são todos eventos de entrada. Os demais são eventos tarefa que atuam em *flags* de saída <*o*>. As cargas conectadas ao sistema (cabeçote de reprodução, cabeçote de gravação e motor) são controladas em função do tratamento dos *flags* de saída <*o*>. Nesse exemplo não existem eventos de sinalização *s* nem *flags* de ação <*a*>. A Figura 7 apresenta um esquema desse sistema.

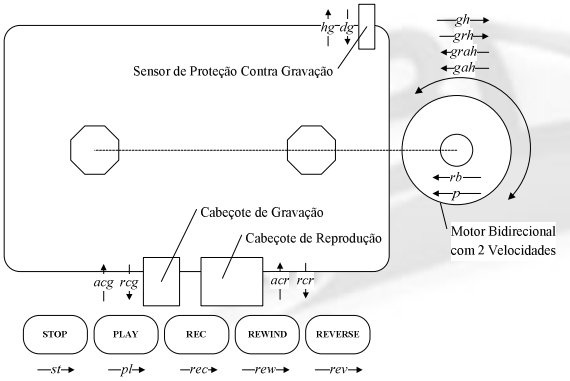

Figura 7. Esquema do Toca-Fitas Auto-Reverso Digital.

Não é possível controlar a ocorrência de pulsos externos nos pinos do microcontrolador, de modo que os eventos de entrada (*st, pl, rec, rew, rev, hg, dg* e *rb*) são de natureza não controlável. Os demais eventos (*acg, rcg, acr, rcr, gh, grh, grah, gah,* e *p*) são tarefas, sendo então considerados controláveis.

O uso de priorização no tratamento dos eventos não controláveis que chegam ao *buffer* B_σ, pode fazer com que eventos controláveis sejam tratados com um pequeno atraso (da ordem de microsegundos) pelo supervisor. A metodologia proposta neste trabalho é adequada para tratar problemas nos quais esse atraso no processamento de eventos controláveis não leva a situações indesejadas. Tendo em vista que o atraso é muito pequeno, uma grande quantidade de problemas reais podem ser solucionados através desta abordagem.

4.1 Descrição da Planta

A planta foi dividida em 5 elementos: cabeçotes de reprodução e de gravação, sensor de proteção contra gravação, motor e teclas.

O modelo dos cabeçotes de reprodução e gravação considera 2 estados: cabeçote retraído (*0*) e acionado (*1*), com o autômato transitando de um estado ao outro na ocorrência dos eventos *acr*, *acg*, *rcr* e *rcg*, conforme apresenta a Figura 8.

Figura 8. Autômatos dos Cabeçotes de Reprodução e Gravação.

O modelo do sensor de proteção contra gravação possui 2 estados: gravação desabilitada (*0*) e habilitada (*1*), com o autômato transitando entre esses estados em função dos eventos *hg* e *dg*, conforme apresentado na Figura 9.

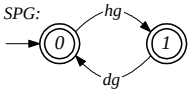

Figura 9. Autômato do Sensor de Proteção Contra Gravação.

O motor foi concebido com 5 estados: parado (*0*), girando horário (*1*), girando rápido horário (*2*), girando rápido anti-horário (*3*) e girando anti-horário (*4*), com os eventos sendo *p*, *gh*, *grh*, *grah* e *gah* respectivamente, como ilustrado na Figura 10.

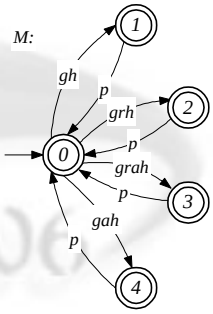

Figura 10. Autômato do Motor.

O modelo das teclas possui os estados parado (*0*), tocando (*1*), gravando (*2*) e rebobinando (*3*). Só é possível acessar o estado *2* a partir do estado *0*. Pode-se alternar entre os estados *1* e *3* sem precisar acessar o estado *0*. O evento reverso (*rev*) só é reconhecido nos estados *1* e *3*. O evento rotor bloqueado (*rb*) possui funções diferentes dependendo do estado em ocorre. No estado *1* esse evento desempenha a função reverso e nos estados *2* e *3* ele realiza a função parar, como ilustra a Figura 11.

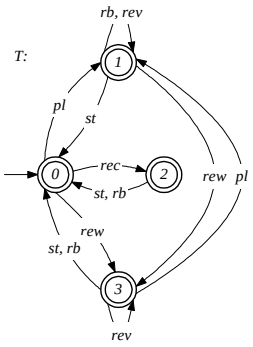

Figura 11. Autômato das Teclas.

A composição síncrona desses elementos gera um autômato com 160 estados e 1256 eventos, que representa o comportamento livre (sem controle) da planta.

4.2 Definição das Especificações

Com o intuito de restringir o comportamento da planta ao comportamento desejado, foram definidas 7 especificações relacionando eventos dos 5 elementos da planta.

A especificação E_1, ilustrada na Figura 12, relaciona o sensor de proteção contra gravação e o motor. Esta especificação habilita a leitura dos eventos do sensor somente com o motor parado.

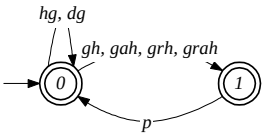

Figura 12. Autômato da Especificação E_1.

A especificação E_2 relaciona os cabeçotes de reprodução e de gravação e o motor, permitindo a ocorrência dos eventos de acionamento e retração dos cabeçotes somente com o motor girando com velocidade normal (no sentido anti-horário ou horário), conforme apresenta a Figura 13.

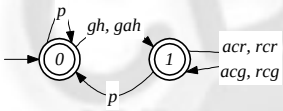

Figura 13. Autômato da Especificação E_2.

A especificação E_3, ilustrada na Figura 14, define o procedimento de retração dos cabeçotes.

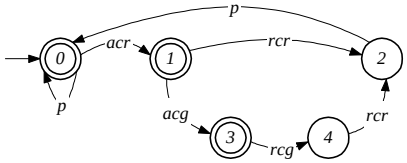

Figura 14. Autômato da Especificação E_3.

A especificação E_4, como mostra a Figura 15, trata do procedimento de acionamento dos cabeçotes.

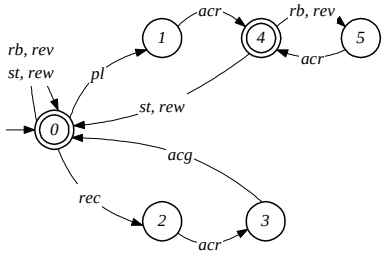

Figura 15. Autômato da Especificação E_4.

A especificação E_5 relaciona os cabeçotes de reprodução e gravação e as teclas, dando ênfase no processo de recolhimento dos cabeçotes. Esta especificação é ilustrada na Figura 16.

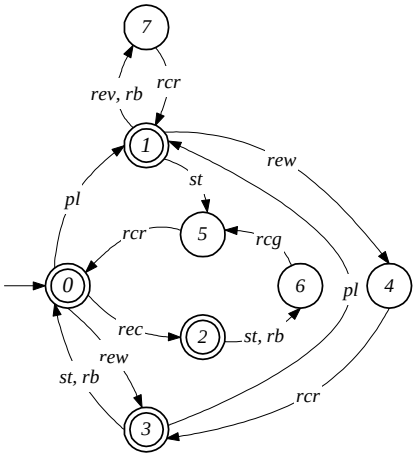

Figura 16. Autômato da Especificação E_5.

A Figura 17 ilustra a especificação E_6, que relaciona o sensor de proteção contra gravação e as teclas, restringindo o acionamento da tecla gravar somente com a gravação habilitada.

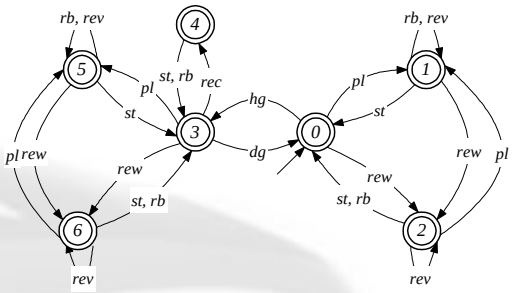

Figura 17. Autômato da Especificação E_6.

Por fim, a especificação E_7, ilustrada na Figura 18, apresenta a relação entre as teclas e o motor, definindo quais tarefas devem ser desempenhadas em função do estado atual e de qual tecla foi pressionada.

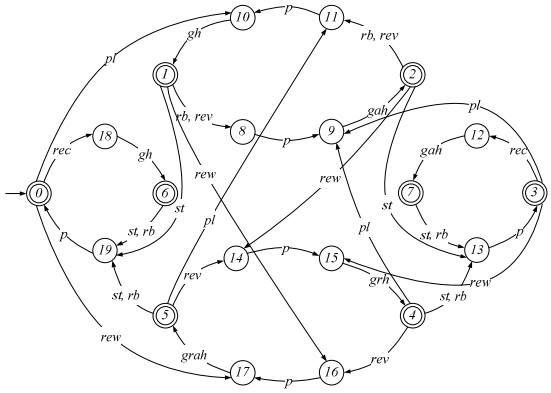

Figura 18. Autômato da Especificação E_7.

4.3 Obtenção do Supervisor

O supervisor foi obtido utilizando o seguinte procedimento: a) cálculo da planta $G = CR \,/\!/\, CG \,/\!/\, SPG \,/\!/\, M \,/\!/\, T$; b) cálculo da especificação global $E = E_1 \,/\!/\, E_2 \,/\!/\, E_3 \,/\!/\, E_4 \,/\!/\, E_5 \,/\!/\, E_6 \,/\!/\, E_7$; c) determinação da linguagem alvo $K = G \,/\!/\, E$; e d) cálculo da máxima linguagem controlável e não bloqueante contida em K, denotada por $supC(G, K) = S$.

O autômato obtido para o supervisor *S* possui 76 estados (sendo 14 marcados) e 108 transições.

A implementação de *S* em linguagem *assembly* para o microcontrolador MICROCHIP PIC16F627A (Microchip Technology Inc., 2005) demandou os recursos apresentados na Tabela 1.

Tabela 1. Recursos Usados na Implementação do Supervisor *S*.

Recurso		**Finalidade**
Tipo	**Uso**	
Memória de dados (224 bytes)	6 bytes	Variáveis
Memória de programa (1024 x 14-bit)	5 words	Inicialização
	13 words	Seleção da tabela
	61 words	Acesso à tabela
	138 words	Tabela T_1
	186 words	Tabela T_2
Total	6 bytes (2,7%) / 403 words (39,4%)	

Note que uma pequena parcela da memória de dados e em torno de 40% da memória de programa são utilizadas, sendo que somente 20% dos 40% de memória de programa ocupados são rotinas de seleção e acesso às tabelas T_1 e T_2, os demais 80% são as tabelas T_1 e T_2 propriamente ditas, as quais contêm a característica do supervisor *S* implementado. Uma única rotina é utilizada para acessar as tabelas T_1 ou T_2.

O uso de 2 tabelas e a demanda de uma rotina de seleção de tabela, são necessários em virtude do processador utilizado permitir somente indexação de 8 *bits*. Como o supervisor possui 108 transições e o tamanho da tabela é 3 vezes o número de transições, seria necessário uma indexação de no mínimo 9 *bits* para viabilizar o uso de uma única tabela. Microcontroladores um pouco maiores já possuem indexação de 16 *bits*, o que melhoraria esse aspecto.

Os dados contidos na Tabela 1 não consideram os recursos necessários para a implementação do *buffer* B_σ, que dependem do número de posições desejado para esse *buffer*.

5 Conclusão

A aplicação da metodologia de implementação proposta na solução do problema Toca-Fitas Auto-Reverso Digital, mostra que a técnica é adequada no tratamento de sistemas embarcados característicos de equipamentos eletrodomésticos produzidos em larga escala, pois ao demandar pequena quantidade de memória de dados e de programa, viabiliza uma solução de baixo custo, característica essencial nesse tipo de produto.

Comparativamente, se for aplicada a técnica de implementação de supervisores proposta por Wood (2005), ao mesmo problema teórico aqui utilizado, seria necessário uma tabela com 1292 elementos (76 estados multiplicado por 17 eventos).

Considerando um mesmo conjunto de eventos (entradas, saídas e sinalizações), modificações nas especificações de controle são implementadas com facilidade, uma vez que basta recalcular o supervisor e atualizar as tabelas relativas a este. Por uma questão de praticidade, macros em EXCEL foram desenvolvidas visando converter as *strings* fornecidas pelo *software* GRAIL, diretamente em código *assembly* no formato de tabela para o microcontrolador MICROCHIP PIC16F627A. A migração para outra plataforma de microcontrolador se mostra simples, visto que é necessário somente implementar as orientações e fluxogramas propostos na seção 3 e modificar o formato de tabela na macro EXCEL visando adequá-lo para a nova plataforma utilizada.

Para problemas mais complexos ou problemas que demandem um controle distribuído, uma abordagem alternativa consiste no uso de supervisores modulares locais (de Queiroz e Cury, 2002), sendo esse o tema a ser tratado na continuidade do presente estudo.

6 Agradecimentos

Os autores agradecem a Embraco Electronic Controls, por incentivar a realização desse trabalho.

7 Referências Bibliográficas

Cassandras, C. G. and Lafortune, S. (1999). *Introduction to Discrete Event Systems*, 2nd Ed, Kluwer Academic Publishers, Massachussetts.

de Queiroz, M. H. and Cury, J. E. R. (2002). Synthesis and Implementation of Local Modular Supervisory Control for a Manufacturing Cell, *6th International Workshop on Discrete Event Systems (WODES)*, v. 1, p. 377-382, Saragoza.

Microchip Technology Inc. (2005). *Datasheet DS40044D, http://www.microchip.com*.

Raymond, D. and Wood, D. (1995). Grail: A C++ library for automata and expressions, *Journal of Symbolic Computation*, 11, pp. 341-350.

Soares, C. (1999). Utilização de Estadogramas na Programação de Microcontroladores, *Jornadas de Engenharia de Telecomunicações e Computadores*, Faculdade de Ciências e Tecnologia, Universidade de Nova Lisboa, Lisboa, Portugal.

Wonham, W. M. (2005). Supervisory Control of Discrete-Event Systems, *Technical Report*, Dept. of Electrical and Computer Engineering, University of Toronto. ECE 1636F/1637S 2005-06, *http://www.control.utoronto.ca/DES*.

Wood, M. M. (2005). *Application, Implementation and Integration of Discrete-Event Systems Control Theory*, Master Thesis, Control of Discrete-Event Systems Lab, Queen´s University, Kingston, Canada.