Ambiente de Desenvolvimento para Aplicações de Controle em Processadores de Pequeno Porte

Marcos V. Linhares, Ricardo B. Borges, Alexandre F. Tondello, Rômulo S. de Oliveira

[1]Departamento de Automação e Sistemas – Universidade Federal de Santa Catarina
UFSC/CTC/DAS – Campus Universitário – Trindade
CEP: 88040-900 – Florianópolis – SC – Brasil

{marcos,romulo,rbb,tondello}@das.ufsc.br

Abstract. *Computing systems can be used to control a large variety of equipaments. Many of these systems are considered embedded systems. The main design tool of a control engineer is the functional block diagram that contains the controller mathematical models. The goal of this work is to retain the characteristics of block diagram design, but also to allow its utilization in a multitask environment, while executing on low cost DSPs. A multitask model is defined, an appropriate microkernel is designed to support it, and a visual tool is developed to integrate all.*

Resumo. *Os sistemas de computação podem ser utilizados para controlar uma grande variedade de equipamentos. Muitos destes sistemas são considerados sistemas embutidos. A ferramenta de projeto do engenheiro de controle são os diagramas de blocos funcionais onde estão contidos os modelos matemáticos do controlador. O objetivo deste trabalho é manter as características do projeto usando diagramas de blocos e possibilitar a sua utilização em um ambiente multitarefa, executando em processadores DSP de pequeno porte. Para tanto, é definido um modelo multitarefa, um núcleo operacional de tempo real e uma ferramenta de programação visual.*

1. Introdução

Nos últimos anos tem-se visto um crescimento extraordinário da indústria eletrônica e um dos principais motivos está na incorporação de sistemas eletrônicos em uma grande variedade de produtos como automóveis, eletrodomésticos e outros. A introdução de sistemas eletrônicos em equipamentos tradicionais tornou-os mais eficientes, de melhor qualidade e mais baratos. Dentre estes componentes eletrônicos pode-se incluir aqueles que permitem algum tipo de computação: os microprocessadores e microcontroladores.

Estes componentes eletrônicos melhoraram bastante a qualidade do produto final mas o projeto dos sistemas tornou-se bem mais complexo já que envolve uma grande quantidade de elementos heterogêneos (componentes analógicos e digitais). Existe uma tendência que os sinais analógicos sejam processados como sinais digitais, isso devido a grande facilidade de integração entre componentes digitais e a grande flexibilidade destes, pois podem envolver algum tipo de computação e controle em seu interior.

Atualmente, desde simples máquinas domésticas até completas instalações de produção são controlados utilizando sistemas computacionais. Estes sistemas são chamados de sistemas embutidos (*embedded systems*)[Berger 2002][Simon 1999] quando estão

tão fixados ao equipamento que, se forem retirados, estes deixam de funcionar. Muitas vezes os sistemas utilizados no controle de um equipamento possuem restrições temporais que devem ser consideradas sob a pena do controle não ser suficientemente eficiente, quando isto ocorre tem-se um sistema de tempo real. Nestes sistemas os resultados produzidos e o instante em que são produzidos é que determinam o funcionamento correto do sistema[Kopetz 1997][Li and Yao 2003][Stankovic and Ramamrithan 1988].

Muitas aplicações de controle com restrições temporais são implementadas pela codificação de grandes programas em linguagem *assembly*, programando *timers*, manipulando em baixo nível dispositivos periféricos, tarefas e prioridades de interrupção. Apesar do código produzido por estas técnicas ser otimizado para uma execução eficiente, esta abordagem tem desvantagens[Buttazzo 2002], tais como: maior esforço de programação, maior complexidade e difícil manutenção.

Para o projeto e implementação de um sistema de controle de tempo real embutido se utiliza de diversos tipos de profissionais que são especialistas cada um em sua área e que usam suas próprias ferramentas. Um sistema simples envolveria um engenheiro de *hardware* para projetar o *hardware* a ser utilizado, um engenheiro de controle para projetar o controlador e definir as restrições temporais e um engenheiro de *software* para implementar o controlador no *hardware* construído.

A interação entre o engenheiro de controle e o de *software* deve ser a mais eficaz possível já que um está projetando o que o outro irá implementar. E, para tanto, é de extrema valia que ambos troquem informações utilizando uma mesma linguagem. A ferramenta de projeto do engenheiro de controle são os diagramas de blocos funcionais onde estão contidos os modelos matemáticos da planta onde se está atuando e do controlador que se está projetando. De forma a facilitar a comunicação entre eles pode-se mapear os diagramas de blocos em diagramas de componentes de *software* onde o engenheiro de *software* pode retirar as informações para implementar o controlador.

O objetivo deste trabalho é criar um ambiente de desenvolvimento que utiliza o mapeamento de diagramas de blocos para componentes de *software* em um modelo multitarefa, integrando tudo através do desenvolvimento de um pequeno núcleo operacional e uma ferramenta de programação visual.

Este artigo descreve o ambiente de desenvolvimento como um todo. O modelo multitarefa e o núcleo operacional já foram explorados mais detalhadamente [Linhares 2004a] e [Linhares 2004b]. Neste artigo destaca-se, principalmente, a ferramenta visual construída.

2. Componentes de *Software*

Para o engenheiro de controle, projetar sistemas utilizando o diagrama de blocos é bastante intuitivo e conhecido, além de existirem diversas ferramentos utilizando este paradigma na concepção de controladores. Mapear estes blocos funcionais em componentes de *software* [Figoli 2001] torna esta metodologia ainda mais atraente criando um canal de comunicação confiável entre o engenheiro de controle e o engenheiro de *software*. Um bloco funcional representa um modelo matemático que realiza cálculos sobre valores de entrada gerando valores de saída (Figura 1). Na transformação do bloco funcional em um componente de *software* as entradas e saídas se tornam atributos do componente e o modelo matemático corresponde a um algoritmo de computação.

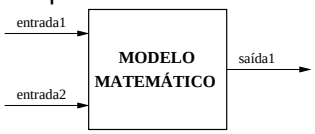

Figura 1. Bloco Funcional genérico.

Seguindo as especificações do *Algorithm Standard*, proposto pela Texas Instruments [Blonstein 2002] [Instruments 2002] para este mapeamento, tem-se como resultado a geração de um algoritmo genérico que poderia ser aplicado a qualquer componente, como pode ser visto abaixo:

```
/* Define a estrutura do COMPONENTE_GENERICO */
typedef struct {
   int entrada1;
   int entrada2;
   int saida1;
   int (*init)();
   int (*update)();
   } COMPONENTE_GENERICO;

/* Define um ponteiro para COMPONENTE_GENERICO */
typedef COMPONENTE_GENERICO  *COMPONENTE_GENERICO_handle;

/* Inicializacao padrao para o objeto COMPONENTE_GENERICO */
#define COMPONENTE_GENERICO_DEFAULTS {NULL,     \
                              NULL,     \
                              NULL,     \
                              (int (*)(int))COMPONENTE_GENERICO_Init,  \
                              (int (*)(int))COMPONENTE_GENERICO_Update \
                              }

/* Prototipos das funcoes */
void COMPONENTE_GENERICO_Init(COMPONENTE_GENERICO_handle);
void COMPONENTE_GENERICO_Update(COMPONENTE_GENERICO_handle);

/* Funcao de inicializacao do Componente */
void inline COMPONENTE_GENERICO_Init(COMPONENTE_GENERICO *p)
{
/* CONFIGURACAO INICIAL */
}

/* Funcao de computacao/atualizacao do Componente */
void inline COMPONENTE_GENERICO_Update(COMPONENTE_GENERICO *p)
{
/* ALGORITMO DE COMPUTACAO */
}
```

A conexão entre os blocos é feita simplesmente ligando a saída de um componente à entrada de outro componente, o código gerado resultante é basicamente a atribuição de uma variável a outra, como pode ser visto a seguir na conexão entre dois componentes genéricos:

```
comp_generico2.entrada1 = comp_generico1.saida;
```

3. Modelo Multitarefa

Um modelo multitarefa é necessário quando existe a necessidade de várias tarefas executando. Este modelo aliado à um suporte adequando, onde as tarefas podem receber prioridades e tempos individuais com requisitos temporais, aumenta-se a capacidade processamento de tarefas por unidade de tempo (*throughput*) além de diminuir os tempos de resposta.

Com o objetivo de aplicar o modelo multitarefa sobre a metodologia de desenvolvimento existente teve-se que criar um elemento que delimita-se as fronteiras entre as tarefas, para isso foi utilizado um retângulo tracejado com cantos arredondados. Cada tarefa está envolvida por este delimitador.

A criação de um sistema embutido de controle é um processo no qual é necessário, também, a interação entre o *hardware* e o *software*. Para tanto, incluiu-se no modelo um elemento para representar o *hardware* externo conectado ao componente de *software*: um retângulo tracejado com cantos aguçados. Isso possibilita ao engenheiro de *software* criar componentes específicos para um determinado dispositivo periférico. Por exemplo, na Figura 2 o bloco "MOTOR" é um exemplo desta classe.

Supondo um sistema com cinco tarefas: controlar um motor e oferecer ao usuário uma interface de comunicação com recebimento e envio de dados via uma interface RS-232 e/ou via uma IHM (interface humano-máquina) composta de um display e um teclado, considerando componentes já desenvolvidos e a simbologia acima especificada, teríamos o sistema multitarefa ilustrado pela Figura 2. Sucintamente as cinco tarefas gerariam o seguinte código genérico, individualmente:

```
void tarefa_generica(void)
{
        while(1)
        {
        /* CODIGO DA TAREFA:
         * - CONEXAO ENTRE OS BLOCOS E SUPORTE OPERACIONAL ADEQUADO
         */
         comp_generico1.update(&comp_generico1);
         comp_generico2.entrada = comp_generico1.saida;
         comp_generico2.update(&comp_generico2);
        }
}
```

Foram definidos dois tipos abstratos de dados de forma a permitir a comunicação entre tarefas, estes elementos foram denominados FILA e ATRIBUTO. A FILA foi criada para ser utilizada quando existe a necessidade de armazenamento de amostragens de um certo parâmetro, geralmente utilizada na comunicação entre uma tarefa de controle e uma tarefa de emissão de dados ou quando existe a necessidade de se possuir o histórico de um determinado parâmetro. É definido por uma estrutura de dados especificada no núcleo de suporte e está representada na Figura 2, onde a entrada é realizada por uma operação

de inserção (*put*) e a saída por uma operação de retirada (*get*) de elementos. Este bloco é baseado em conceito de *buffer* circular do tipo FIFO (*First In First Out*). Já o ATRIBUTO tem o objetivo de armazenar parâmetros de configuração ou amostragens em que somente o último valor interessa. Possui uma representação gráfica similar ao tipo FILA mas, diferentemente dele, no tipo ATRIBUTO as entradas realizam operações de atribuição (*set*) e as saídas operações de leitura (*get*) de elementos (Figura 2).

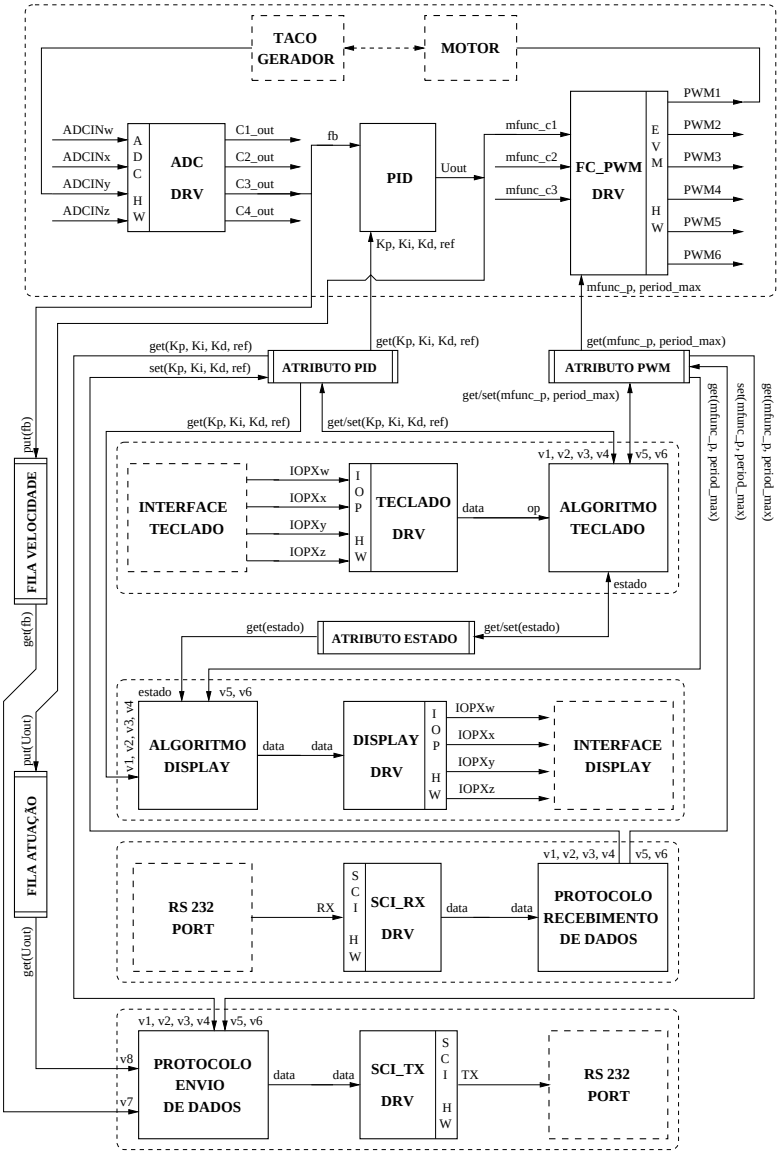

Figura 2. Modelo Multitarefa.

4. O μKernel

Como suporte para o desenvolvimento de aplicações de controle embutidas em DSP de pequeno porte, foi desenvolvido um pequeno núcleo operacional multitarefa de tempo real, denominado μKernel [Linhares 2004a] [Linhares 2004b]. É importante notar que o μKernel foi construído de forma a suportar o ambiente de desenvolvimento explicado nas seções anteriores.

Foi necessário implementar um *microkernel* de tempo real (μKernel) que possuísse os serviços essenciais de forma a não sobrecarregar o processador e que assim fosse pequeno (poucas linhas de código) o suficiente para não ocupar muita memória na plataforma alvo. Existiu, também, a preocupação de que o μKernel pudesse receber componentes desenvolvidos conforme o *Algorithm Standard* da TI, que abrange técnicas para a padronização de codificação de elementos de forma a poderem ser intercambiáveis entre desenvolvedores.

Levando em consideração as restrições envolvidas no desenvolvimento do μKernel, estipulou-se os serviços essenciais a serem prestados por este:

- Gerenciamento de tarefas - serviços disponibilizados para manutenção e execução das tarefas: a) criação de tarefas; b) escalonamento e despacho de tarefas; c) chaveamento de contexto.
- Manipulação do tempo - serviço que trata das requisições de interrupção geradas pelo *timer* do DSP.
- Comunicação entre tarefas - serviços de troca de dados ou configuração de componentes: a) tipo abstrato FILA; b) tipo abstrato ATRIBUTO.

5. Ferramenta de Programação Visual

A programação manual, de maneira geral, é demorada e está sujeita a erros. Essa tendência cresce à medida que as estratégias de controle empregadas tornam-se cada vez mais complexas. Para complementar a criação do ambiente de desenvolvimento, foi desenvolvido uma ferramenta de programação visual para a configuração do software do controlador embutido a partir de diagramas de blocos funcionais. A interface do programa e o exemplo de um diagrama de blocos da ferramenta estão ilustrados na figura 3. Dessa forma é possível projetar o algoritmo de controle através de diagrama de blocos, gerar código automaticamente e transferir este para execução no hardware do controlador.

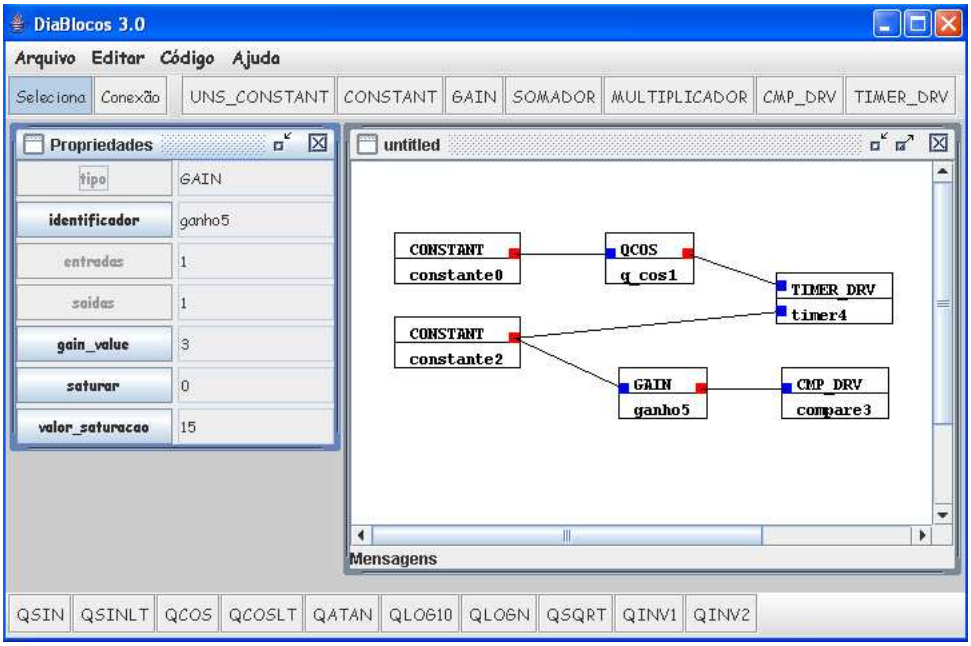

Figura 3. Ferramenta de programação visual.

O código gerado segue a sintaxe genérica descrita anteriormente, além de incluir trechos de código referentes a inicialização do *hardware* alvo, cabeçalhos e a criação e inicialização dos blocos. A inicialização dos blocos é feita a partir dos parâmetros definidos pelo usuário na ferramenta durante a composição da tarefa. A figura 4 descreve o conjunto de ações que podem ser realizadas pelo usuário no sistema.

Figura 4. Casos de uso da ferramenta.

A vantagem da ferramenta desenvolvida sobre outras, como o *Embedded Target* do *Simulink* [MathWorks 2005], é que ela gera somente o código de cola entre os componentes, o que o torna mais legível e transparente para o usuário. A implementação prévia dos blocos é feita pelo especialista na plataforma e o usuário tem acesso a cada um dos arquivos referentes aos blocos, podendo alterar a implementação dos mesmos ou então criar e utilizar novos blocos.

6. Conclusões

Este trabalho descreveu uma proposta que visa melhorar o processo de construção (projeto e implementação) de sistemas embutidos multitarefa em processadores de pequeno porte. Para tanto, o trabalho consistiu da proposta de um modelo multitarefa, o desenvolvimento de um núcleo operacional de tempo real e a criação de uma ferramenta visual para dar suporte ao ambiente desenvolvido.

Como forma de dar suporte ao modelo multitarefa proposto desenvolveu-se um núcleo operacional, chamado de μKernel. O projeto do μKernel foi influenciado, principalmente, pelo fator simplicidade (incluir o mínimo necessário) e confiabilidade (empregar técnicas simples). Por exemplo, qualquer *device driver* aparece como um bloco no projeto da aplicação e seu algoritmo de tratamento passa a fazer parte, principalmente, da tarefa onde está inserido.

Para facilitar o desenvolvimento de aplicações multitarefa, neste ambiente proposto, desenvolveu-se uma ferramenta de programação visual pela composição de blocos

funcionais, com a geração de um código de 'cola' de fácil manutenção e compreensão pelo usuário final. O objetivo é manter cada indivíduo envolvido com a tarefa de seu domínio específico, aumentando a produtividade destes em suas áreas de formação.

Espera-se com este trabalho contribuir para a melhoria dos processos de desenvolvimento de *software* para sistemas embutidos no contexto do controle e automação. Especialmente aqueles de pequeno porte como controladores de motores elétricos e inversores. O ambiente proposto preserva o diagrama de blocos como linguagem do engenheiro de controle, mas permite a inclusão de funcionalidades variadas através do multitarefa, suportado por um μKernel contruído sob medida e a facilidade da programação através de uma ferramenta visual.

7. Agradecimentos

Agradecemos ao CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico pelo financiamento parcial deste trabalho.

Referências

Berger, A. S. (2002). *Embedded Systems Design: An Introduction to Process, Tools and Techniques*. CMP Books.

Blonstein, S. (2002). *The TMS320 DSP Algorithm Standard*. Technical Document - White Paper. Texas Instruments. revision C.

Buttazzo, G. C. (2002). *Hard real-time computing systems: predictable scheduling algorithms and applications*. The Kluwer international series in engineering and computer science. Real-time systems. Kluwer Academic Publishers, 4 edition.

Figoli, D. (2001). *A Software Modularity Strategy for Digital Control Systems*. Technical Document - Application Report. Texas Instruments, DCSA.

Instruments, T., editor (2002). *TMS320 DSP Algorithm Standard - Rules and Guidelines*. Technical Document - User Guide. Texas Instruments. revision E.

Kopetz, H. (1997). *Real-time systems: design principles for distributed embedded applications*. Kluwer Academic Publishers.

Li, Q. and Yao, C. (2003). *Real-time concepts for embedded systems*. CMP Books.

Linhares, M. V. (2004a). *Modelo de programação e suporte de execução para aplicações multitarefa em processadores DSP de pequeno porte*. Dissertação de mestrado, Universidade Federal de Santa Catarina, UFSC, Florianópolis, Brasil.

Linhares, M. V. (2004b). A proposal for the conception of multitask embedded control systems target to low-end dsps. In *Anais do VI INDUSCON - Conferência Internacional de Aplicações Industriais*, Joinville, SC.

MathWorks (2005). *http://www.mathworks.com*.

Simon, D. E. (1999). *An Embedded Software Primer*. Addison-Wesley.

Stankovic, J. and Ramamrithan, K., editors (1988). *Tutorial on Hard Real-Time Systems*. IEEE Computer Society Press.