RTKPIC18

(Real-Time Kernel PIC18FXX8)

Breve Manual

Pedro Leite nº21526
Ricardo Marau nº21089

Janeiro 2004

1. Introdução

O *kernel* RTKPIC18 foi desenvolvido no âmbito de um projecto da cadeira de Sistemas de Tempo Real, do Departamento de Electrónica e Telecomunicações da Universidade de Aveiro, no ano lectivo de 2003/2004, 1º semestre. Este é um *kernel* desenvolvido para os microcontroladores da família PIC18FXX8 e escrito em linguagem C e compilado usando o PICC-18 v8.20PL4 da HI-TECH Software (http://www.htsoft.com). Todo o desenvolvimento do *kernel*, desde da fase de projecto, implementação, teste, *debug* e simulação foi efectuado recorrendo a um IDE da HI-TECH Software, o HI-TIDE v1.0PL4 (HI-TECH *Integrated Development Environment*). Este é um *software* de desenvolvimento grátis que pode ser encontrado na referida página de internet.

Este é um *kernel* multitarefa, preemptivo e com preocupações de tempo-real. Permite definir tarefas periódicas, com relação de fase e deadline, que são activadas automaticamente de forma transparente para utilizador. O *kernel* efectua um escalonamento de tempo-real baseado em prioridades fixas ou prioridades dinâmicas. Para prioridades fixas utiliza *Rate Monotonic* (prioridade inversamente proporcional ao período) ou *Deadline Monotonic* (prioridade inversamente proporcional à deadline). Para prioridades dinâmicas utiliza *Earliest Deadline First* (prioridade proporcional à proximidade das *deadlines* em *runtime*).

O *kernel* nunca foi testado em controlo de sistemas pelo que os robôs do concurso Microrato são boas plataformas de teste ao bom funcionamento, fiabilidade e robustez do *kernel*. Este documento visa apresentar o *kernel* na sua vertente mais funcional, apresentando todas a funções e macros disponíveis e indicando quais as restrições impostas ao utilizador pelo uso do *kernel*.

2. A livraria rtkpic18.obj

A livraria rtkpic18.obj contém as funções do *kernel* multitarefa, preemptivo e de tempo-real para a família PIC18FXX8. Esta facilita a programação de um sistema de controlo baseado em tarefas independentes e periódicas com restrições temporais. Basicamente, este *kernel* permite transformar normais funções de C em tarefas periódicas cuja activação e execução concorrente é controlada pelo próprio *kernel* e completamente transparente para o utilizador. Com o código do utilizador e esta livraria é gerado um código monolítico, que depois de compilado e assemblado está pronto a ser carregado para o microcontrolador.

Toda a configuração do sistema, desde resolução temporal do *kernel*, algoritmo de escalonamento, activação da preempção, assim como a criação de tarefas é feita *offline*, sendo depois disparado entrando em execução. Para a criação de tarefas é necessário fornecer parâmetros descritivos como o período, instante da primeira activação e *deadline*. O *kernel* responsabilizar-se-á de atribuir prioridades fixas (DM ou RM) ou dinâmicas (EDF) às tarefas consoante a configuração requerida pelo utilizador.

Sempre que uma tarefa de maior prioridade fica activa durante a execução de outra de menor prioridade esta última é interrompida de modo a que a primeira possa usar o CPU (preempção). Após a terminação da tarefa de maior prioridade a de menor prioridade reata a sua execução no ponto de interrupção. Assim, note-se que a tarefa de maior prioridade no sistema nunca é interrompida mas as restantes podem ser. A utilização ou não de preempção é da responsabilidade do utilizador na fase de configuração.

Este *kernel* não possuí protocolos de acesso a recursos partilhados pelo que a comunicação poderá ser feita recorrendo a variáveis globais com inibição de interrupções para acesso à região crítica (no entanto a inibição das interrupções deverá ser muito curta para não afectar o controlo temporal efectuado pelo *kernel*).

2.1. Funções disponíveis

signed char create_system(unsigned int tick_ms, unsigned char task_sch, unsigned char preempt);

Cria um novo sistema com as definições configuradas pelo utilizador: período do *tick* do sistema, algoritmo de escalonamento e preempção.

O parâmetro tick_ms especifica a duração em milisegundos do *tick* do sistema. Este parâmetro é validado com um valor entre 2 e 65534 (múltiplos de 2). O controlo temporal do *kernel* é efectuado com o uso do *timer2* pelo que este não poderá ser utilizado por nenhuma tarefa.

O parâmetro task_sch define o algoritmo de escalonamento de tarefas a ser utilizado pelo *kernel*. São validados os valores RM (*Rate Monotonic*), DM (*Deadline Monotonic*) e EDF (*Earliest Deadline First*).

Finalmente o parâmetro preempt especifica se o sistema vai ser ou não preemptivo. Os valores PREEMPT (para activar a preempção) e NOPREEMPT (para não activar a preempção) são validados.

Esta função retorna OK se o sistema foi criado com sucesso ou um erro reportando o que correu mal: TICK_ERR (erro na configuração do *tick*), SCHEDULER_ERR (erro na

configuração do algoritmo de escalonamento) ou PREEMPT_ERR (erro na configuração da preempção).

signed char create_task(void (*func_ptr)(void), unsigned int period, unsigned int first, unsigned int deadline);

Cria uma tarefa deixando-a no estado IDLE à espera da sua activação periódica. É programada um entrada no chamado TCB (*Task Control Block*) onde ficam guardadas todas as propriedades das tarefas, desde período, *deadline*, estado, prioridade, entre outras.

O parâmetro de entrada *func_ptr indica o endereço da primeira instrução da tarefa, ou seja, por outras palavras deve ser atribuído o nome da tarefa periódica.

O parâmetro period define o período de activação da tarefa.

O parâmetro first especifica o instante da primeira activação.

Por fim o parâmetro deadline define a *deadline* relativa a uma activação na qual a execução da tarefa deve estar concluída.

Estes parâmetros temporais devem ser indicados em *ticks*, ou seja, em múltiplos inteiros do *tick* definido na função create_system.

O valor devolvido pela função é o identificador da tarefa ($1 \leq id \leq 13$) ou TASK_CREAT_ERR se a tarefa não pôde ser criada.

void config_system(void);

Após as especificações do sistema e das tarefas, o *kernel* irá configurar o sistema utilizando para isso a função config_system. Nesta função são inicializadas variáveis globais como o relógio do sistema (system_clock), flags de controlo de falha de *deadlines* (deadline_miss). São atribuídas prioridades no caso de prioridades fixas e o *timer* utilizado (*timer2*) pelo sistema é activado. O sistema após a execução desta função está pronto a ser disparado.

void task_init(void);
void end_cycle(void);

Estas são funções especiais que devem ser utilizadas em todas as tarefas. Uma tarefa deve seguir um modelo de ciclo de execução:

```
void nome_tarefa(void)
{
        (...)     /* definição de variáveis (não inicializar) */

        task_init();

        (...)     /* inicialização de variáveis estáticas */

        while(1)
        {
                (...)     /* inicialização de variáveis locais */
                (...)     /* código periódico */
```

```
            end_cycle();
        }
    }
```

A função task_init é executada uma única vez e é utilizada para inicializar a zona de salvaguarda de contexto. A função end_cycle é uma chamada ao *kernel* que indica que um ciclo da tarefa acabou passando esta para o estado IDLE à espera de uma nova activação.

2.2. Macros disponíveis

void release_system()

Como foi dito as tarefas após criação ficam no estado IDLE. O sistema é configurado ficando à espera de ser disparado. Esta macro release_system permite o disparo do sistema, ou seja, todas as tarefas periódicas entram em execução sincronamente, o que quer dizer que se inicia a contagem do tempo (em *ticks*) por parte do sistema para a primeira activação.

long get_sys_time()

Esta macro devolve o número de *ticks* que ocorreram desde o início do funcionamento do sistema. Pode ser utilizada para medições temporais (em *ticks*) quer absolutas quer relativas calculando a diferença entre o respectivo valor em dois instantes diferentes.

char get_my_id()

Esta macro devolve o identificador da tarefa que a invoca. Os identificadores são atribuídos pela função create_task sequencialmente, a partir do identificador 1 (o *main* ou tarefa de *background*, tem sempre o identificador reservado 0).

char get_deadl_stat(id)

Esta macro devolve o estado da tarefa identificada por id, em termos de prazo de execução (*deadline*). Se a tarefa ainda não ultrapassou nenhum prazo é devolvido 0. Se a tarefa se atrasou para além do prazo (pelo menos uma vez) então esta macro devolve o valor 1.

void recursive_call()

Esta macro deverá aparecer no código fonte da aplicação aquando da alusão das tarefas periódicas. Esta macro é responsável pela não sobreposição das variáveis das tarefas e por isso é fundamental.

2.3. Utilização do *kernel* RTKPIC18

Para utilizar o *kernel* rtkpic18 deverá ter em atenção o seguinte:

- não usar o *timer2* do microcontrolador;

· em geral, não inibir as interrupções durante a execução das tarefas já que isso levaria a erros na contagem temporal do sistemas com consequências inerentes a um sistema de tempo-real. Contudo, a inibição pode ser utilizada durante pequenos intervalos, por exemplo para garantir o acesso atómico a estruturas (ou variáveis) partilhadas;

· incluir o ficheiro rtkpic18.h com as definições e declarações necessárias à utilização do sistema;

· escrever o código das tarefas seguindo o modelo descrito. A repetição periódica das tarefas é completamente controlada pelo sistema;

· por seu lado, o *main* deverá começar por chamar a função create_system, criar as tarefas necessárias com a função create_task, configurar o sistema com a função config_system e disparar o sistema com a macro release_system entrando depois num ciclo infinito tipo while(1). O que for executado neste ciclo do *main* diz-se que corre em *background*, quando não existem tarefas prontas a executar;

· para não aumentar demasiado o tempo de execução das tarefas dever-se-á ter o cuidado de não efectuar bloqueios dentro destas, como por exemplo, esperar pela recepção ou transmissão de um caracter pela porta série, ou fazer um ciclo infinito.

3. Geração de código

A aplicação de controlo escrita em linguagem C pelo utilizador deverá ser compilada e linkada com a livraria rtkpic18.obj usando o compilador pago PICC-18 da HI-TECH Software. Para isso o utilizador poderá utilizar a *makefile* disponível (mk_rtk18.bat) que aceita como parâmetro o nome do programa e gera um ficheiro no formato Intel HEX pronto a ser carregado no microcontrolador. Ainda são gerados outros ficheiros como o ficheiro *assembly*, o mapeamento da memória, etc.

Exemplo: exemple.c → mk_rtk18 exemple → exemple.hex

Esta *makefile* utiliza algumas opções do compilador (dispositivo: 18F258, optimizações, etc.) pelo que é aconselhado ao utilizador editá-la e alterá-la consoante as suas preferências. As opções do compilador estão no ficheiro PICC18 options.txt.

Outra alternativa (e mais do que recomendada) é a utilização do software grátis de desenvolvimento da HI-TECH Software, o HI-TIDE. Este *software* apresenta um ambiente gráfico, possuí um editor e possibilita a simulação da aplicação compilada. Possuí bastantes ferramentas de auxílio ao *debug* como execução passo a passo, visualização da memória, *breakpoints*, visualização de variáveis, I/O virtual, entre outras. O utilizador apenas precisa de criar um novo projecto, alterar as opções globais e incluir a livraria rtkpic18.obj assim como o código escrito para a aplicação. Depois basta compilar e linkar fazendo a parte de teste e *debug* neste ambiente de desenvolvimento sem necessidade do microcontrolador.