CCS リアル・タイム・オペレーティング・システム(RTOS)

CCS リアル・タイム・オペレーティング・システム(RTOS) は PIC マイクロ・コントローラーがインターラプトを必要としないで周期的にスケジュールされたタスクを実行することが出来る OS です。これはディスパッチャーとして動作する関数(RTOS_RUN()) によって行われます。タスクの実行が予定されますと、ディスパッチ関数がそのタスクにプロセッサーからの制御を与えます。タスクが実行される時やプロセッサがディスパッチ関数から戻って来てそれ以上のタスク実行が無い場合、プロセッサの制御は次のタスクとして予定された専有時間を実行します。このプロセスは協調マルチタスキングと呼ばれます。

#USE RTOS

構文: #use rtos(***options***)

エレメント: ***options*** : はカンマ(,)でセパレートされ、そして、下記の指定が可能です。

timer=X	x は 0-4 です。RTOS により使用されるタイマーを指定します。
minor_cycle=time	time は s, ms, us, ns の付いた数字です。これは全てのタスクで実行出来る最長時間です。各タスクの実行レートはこの時間の倍数でなければいけません。もし、指定されていなくても、コンパイラは各々のタスクで使用する最小、最大、合計時間を計算します。
statistics	各々のタスクで使用する最小、最大、合計時間

目的: このディレクティブはコンパイラにタスクの制御を許可する時、PIC のどのタイマーを使用してモニターするのかを伝えます。指定したタイマーのプリスケーラが変化すると、タスクの実行されるレートが変わります。このディレクティブは minor_cycle のオプションで、タスクにかかっている最も長い時間を特定することができます。

またこのディレクティブはタスクを minor_cycle オプションで実行するのに最長時間を指定するために使用することが出来ます。
これは単にプロジェクトがコンパイルを完了する前に、タスクの実行レートを minor_cycle の倍数に強制的に設定します。もし、このオプションが設定されていない場合、コンパイラは RTOS タスクの実行レートから使用する minor_cycle 値を最も可能な最小係数に設定します。

スタティクス・オプションが指定されますと、コンパイラーは各タスクの各々の実行に掛かる最小と最大のプロセッサー時間、及び各々のタスクで使用する合計のプロセッサー時間を追跡します。

サンプル: #use rtos(timer=0, minor_cycle=20ms)

参照: #task

各 RTOS タスクはパラメータと戻り値を持たない関数として指定されます。
#task ディレクティブはコンパイラーにどの関数が RTOS タスクであるかを伝えるためにイネーブルにする必要が有りますので、各 RTOS タスクの前に必要です。

#TASK

構文: #task(***options***)

エレメント: ***options*** はカンマ(,)でセパレートされ、そして、下記の指定が可能です。

rate=time	time は s，ms，us，又は、ns の付いた数字です。これはどの周期でタスクが実行されるかを指定します。
max=time	time は s，ms，us，又は、ns の付いた数字です。これはタスクに割り当てた時間を指定します。
queue=bytes	タスクの入力メッセージに必要なバイト数を指定します。デフォルトは 0 です。

目的: このディレクティブはコンパイラに下記の関数が RTOS タスクである事を伝えます。

rate オプションはタスクが何度実行されるかを指定するのに使用されます。これは#user rtos ディレクティブで指定されている場合は、minor_cycle オプションの倍数でなければいけません。

max オプションは 1 つのタスクの実行で、プロセッサー時間がいくら使用されるかを指定するのに使われます。max で指定された時間は、プロジェクトのコンパイルが完了する前に#use rtos ディレクティブの minor_cycle オプションで指定された時間と同じか、若しくは、少なくなければいけません。

コンパイラーはプロセッサー時間についてこの制限内で強制する方法は有りませんので、タスクが実行に使用するプロセッサー時間をいくらにするか注意して下さい。このオプションを指定する必要はありません。

queue オプションは他のタスク、又は関数から、そのタスクがメッセージを受信するために確保されるバイト数の指定に使用されます。queue のデフォルトは 0 です。

サンプル:#task(rate=1s, max=20ms, queue=5)

参照: #use rtos

RTOS_RUN()

構文: rtos_run()
パラメータ: なし
戻り値: なし
機能: この関数は各タスクの割り当てられた比率で、RTOS タスクの実行を制御します。この関数は RTOS_TERMINATED()がコールされた時にのみリターンします。
対象デバイス: 全デバイス
要求事項: #use rtos
サンプル: rtos_run();
参照: RTOS_TERMINATE()

RTOS_TERMINATE()

構文: rtos_terminate()
パラメータ: なし
戻り値: なし
機能: この関数はすべての RTOS タスクの実行を終了します。プログラムの実行は、プログラムから RTOS_RUN()をコールした後の最初のコード行を続行します。(この関数で RTOS_RUN()はリターンとなります。)
対象デバイス: 全デバイス
要求事項: #use rtos
サンプル: rtos_terminate();
参照: RTOS_RUN()

RTOS_ENABLE()

構文: rtos_enable(***task***);
パラメータ: ***task*** は RTOS タスクとして使用される関数の識別子です。
戻り値: なし
機能: この関数は指定されたレートでタスクをイネーブルにします。
すべてのタスクはデフォルトでイネーブル(有効)になっています。
対象デバイス: 全デバイス
要求事項: #use rtos
サンプル: rtos_enable(toggle_green);
参照: RTOS_DISABLE()

RTOS_DISABLE()

構文: rtos_disable(***task***)
パラメータ: ***task*** は RTOS タスクとして使用される関数の識別子です。
戻り値: なし
機能: この関数は RTOS_ENABLE でタスクを有効にするまで、ディスエーブル(無効)にします。すべてのタスクはデフォルトでは、イネーブルとなっています。
対象デバイス: 全デバイス
要求事項: #use rtos
サンプル: rtos_disable(toggle_green);
参照: RTOS_ENABLE()

RTOS_MSG_POLL()

構文: i = rtos_msg_poll()
パラメータ: なし
戻り値: いくつのメッセージがキュー(処理を待っている状態)にあるかを表す整数です。
機能: この関数は RTOS タスクにのみ使用されます。この関数は RTOS_MSG_POLL 関数が使われているタスクに対して、キューにあるメッセージの数を返します。
対象デバイス: 全デバイス
要求事項: #use rtos
サンプル: if(rtos_msg_poll())
参照: RTOS_MSG_SEND(), RTOS_MSG_READ()

RTOS_MSG_READ()

構文: b = rtos_msg_read()

パラメータ:　　なし
戻り値: バイトで、タスクへのメッセージです。
機能:　　この関数は RTOS タスクにのみ使用されます。この関数は RTOS_MSG_READ 関数が使われているタスクのキューにある次のメッセージを読み込みます。
対象デバイス: 全デバイス
要求事項: #use rtos
サンプル:
```
if(rtos_msg_poll()) {
        b = rtos_msg_read();
```
参照:　　RTOS_MSG_POLL(), RTOS_MSG_SEND()

RTOS_MSG_SEND()

構文:　　rtos_msg_send(***task, byte***)
パラメータ:　***task*** は RTOS タスクとして使用される関数の識別子です。
　　byte は、メッセージである ***task*** へ送るバイトです。
戻り値: なし
機能:　　この関数は RTOS_RUN()がコールされた後、いつでも使用することが出来ます。この関数は ***task*** により認識されたタスクへ、バイト長のメッセージ(***byte***)を送ります。
対象デバイス: 全デバイス
要求事項:　　#use rtos
サンプル:
```
if(kbhit())
    {
        rtos_msg_send(echo, getc());
    }
```
参照:　　RTOS_MSG_POLL(), RTOS_MSG_READ()

RTOS_YIELD()

構文:　　rtos_yield()
パラメータ:　　なし
戻り値:　なし
機能:　　この関数は RTOS タスクにのみ使用されます。この関数は現在のタスクの実行をストップし、そして、プロセッサーの制御を RTOS_RUN に戻します。次にタスクを実行するのは、RTOS_YIELD の後のコード行でスタートします。
対象デバイス: 全デバイス
要求事項: #use rtos
サンプル:
```
void yield(void)
    {
        printf("Yielding...\r\n");
        rtos_yield();
        printf("Executing code after yield\r\n");
    }
```
参照:　　なし

RTOS_SIGNAL()

構文:　　rtos_signal(***sem***)

パラメータ: *sem* は現在利用可能な共有システム･リソース(セマフォ)を表すグローバル変数です。

***セマフォ[system resource (a semaphore).]はリソースに対するアクセスを調整するメカニズムを提供します。どのようなものにもアクセス制御として利用出来ます。**

戻り値: なし
機能:　　この関数は RTOS タスクにのみ使用されます。この関数はタスクが利用可能な共有リソースを知るまで待たせるため、sem を減少します。
対象デバイス: 全デバイス
要求事項: #use rtos
サンプル:
```
rtos_signal(uart_use);
```
参照:　　RTOS_WAIT()

RTOS_WAIT()

構文:　　rtos_wait(***sem***)
パラメータ: ***sem*** は現在利用可能な共有システム・リソース(セマフォ)を表すグローバル変数です。
戻り値: なし

機能:　　この関数は RTOS タスクにのみ使用されます。この関数は ***sem*** が 0(共有リソースが利用可能)になるまで待ち、そして共有リソースの使用を要求するために ***sem*** を増加させます。そして RTOS タスクは残りのコードの実行を続行します。この関数はあるタスクが共有リソースの利用を待っている間、他のタスクが実行することを許可します。
対象デバイス: 全デバイス
要求事項: #use rtos
サンプル: rtos_wait(uart_use);
参照:　　RTOS_SIGNAL()

RTOS_AWAIT()

構文:　　rtos_await(***expre***)
パラメータ: ***expre*** は論理式です。
戻り値: なし
機能:　　この関数は RTOS タスクでのみ使用することが出来ます。
この関数は ***expre*** が真になるまで、RTOS タスクの残りのコードの実行を待ちます。この関数は ***expre*** が真になる間、他のタスクの実行を許可します。
対象デバイス: 全デバイス
要求事項: #use rtos
サンプル: rtos_await(kbhit());
参照:　　なし

RTOS_OVERRUN()

構文:　　rtos_overrun(***[task]***);
パラメータ: ***task*** は RTOS タスクで使用される関数の識別子としてのオプションのパラメータです。
戻り値: A 0 (FALSE) 又は、1 (TRUE)
機能:　　この関数は指定されたタスクの実行が割り当てられた時間以上掛かるときに TRUE を返します。もし、タスクが指定されていない場合は、どれかひとつでもタスクの割り当てられた実行時間を越えると TRUE を返します。
対象デバイス: 全デバイス
要求事項: #use rtos(statistics)
サンプル: rtos_overrun();
参照:　　なし

RTOS_STATS()

構文:　　rtos_stats(***task,stat***)
パラメータ: ***task*** は RTOS タスクとして使用されるべき関数の識別子です。
　　stat は下記の 1 つです。:
　　rtos_min_time – 指定された ***task*** の 1 つの実行に必要な最小の処理時間
　　rtos_max_time –指定された ***task*** の 1 つの実行に必要な最大の処理時間
　　rtos_total_time – タスクに使用される合計の処理時間
戻り値:　32 ビット整数が us 単位で表され指定されたタスク(***task***)に対して、スタート(***stat***)で指定されます。
機能:　　この関数は指定された ***task***.に対して指定された ***stat*** を返します。
対象デバイス: 全デバイス
要求事項: #use rtos(statistics)
サンプル: rtos_stats(echo, rtos_total_time);
参照:　　なし

下記のデモ・サンプル・ファイルは **PIC18F452 ミニプロトタイプ・ボード**を使用して作成されました。

```
////////////////////////////////////////////////////////////////////////////////////////////////////
// This file demonstrates how to use the real time operating system        //
// to schedule tasks and how to use the rtos_run function                  //
//                                                                         //
// this demo makes use of the PIC18F452 prototyping board                  //
////////////////////////////////////////////////////////////////////////////////////////////////////

#include <18F452.h>
#device ICD=TRUE
#fuses HS,NOLVP,NOWDT,PUT
#use delay(clock=20000000)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7)
```

```
// this tells the compiler that the rtos functionality will be needed, that
// timer0 will be used as the timing device, and that the minor cycle for
// all tasks will be 500 miliseconds
#use rtos(timer=0,minor_cycle=100ms)
// each function that is to be an operating system task must have the #task
// preprocessor directive located above it.
// in this case, the task will run every second, its maximum time to run is
// less than the minor cycle but this must be less than or equal to the
// minor cycle, and there is no need for a queue at this point, so no
// memory will be reserved.
#task(rate=1000ms,max=100ms)
// the function can be called anything that a standard function can be called
void The_first_rtos_task ( )
{
    printf("1¥n¥r");
}
#task(rate=500ms,max=100ms)
void The_second_rtos_task ( )
{
    printf("¥t2!¥n¥r");
}
#task(rate=100ms,max=100ms)
void The_third_rtos_task ( )
{
    printf("¥t¥t3¥n¥r");
}
// main is still the entry point for the program
void main ( )
{
    // rtos_run begins the loop which will call the task functions above at the
    // schedualed time
    rtos_run ( );
}


////////////////////////////////////////////////////////////////////////////////////////////////////
// This file demonstrates how to use the real time operating system        //
// rtos_terminate function                                                 //
//                                                                         //
// this demo makes use of the PIC18F452 prototyping board                  //
////////////////////////////////////////////////////////////////////////////////////////////////////

#include <18F452.h>
#device ICD=TRUE
#fuses HS,NOLVP,NOWDT,PUT
#use delay(clock=20000000)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7)
#use rtos(timer=0,minor_cycle=100ms)
// a counter will be kept
int8 counter;
#task(rate=1000ms,max=100ms)
void The_first_rtos_task ( )
{
    printf("1¥n¥r");
    // if the counter has reached the desired value, the rtos will terminate
    if(++counter==5)
        rtos_terminate ( );
```

```
}
#task(rate=500ms,max=100ms)
void The_second_rtos_task ( )
{
    printf("¥t2!¥n¥r");
}
#task(rate=100ms,max=100ms)
void The_third_rtos_task ( )
{
    printf("¥t¥t3¥n¥r");
}
void main ( )
{
    // main is the best place to initialize resources the the rtos is dependent
    // upon
    counter = 0;
    rtos_run ( );
    // once the rtos_terminate function has been called, rtos_run will return
    // program control back to main
    printf("RTOS has been terminated¥n¥r");
}


////////////////////////////////////////////////////////////////////////////////////////////////////
// This file demonstrates how to use the real time operating system    //
// rtos_enable and rtos_disable functions                              //
//                                                                     //
// this demo makes use of the PIC18F452 prototyping board              //
////////////////////////////////////////////////////////////////////////////////////////////////////

#include <18F452.h>
#device ICD=TRUE
#fuses HS,NOLVP,NOWDT,PUT
#use delay(clock=20000000)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7)
#use rtos(timer=0,minor_cycle=100ms)
int8 counter;
// now that task names will be passed as parameters, it is best
// to declare function prototypes so that their are no undefined
// identifier errors from the compiler
#task(rate=1000ms,max=100ms)
void The_first_rtos_task ( );
#task(rate=500ms,max=100ms)
void The_second_rtos_task ( );
#task(rate=100ms,max=100ms)
void The_third_rtos_task ( );
void The_first_rtos_task ( ) {
    printf("1¥n¥r");
    if(counter==3)
    {
        // to disable a task, simply pass the task name
        // into the rtos_disable function
        rtos_disable(The_third_rtos_task);
    }
}
void The_second_rtos_task ( ) {
    printf("¥t2!¥n¥r");
```

```
    if(++counter==10) {
        counter=0;
        // enabling tasks is similar to disabling them
        rtos_enable(The_third_rtos_task);
    }
}
void The_third_rtos_task ( ) {
    printf("¥t¥t3¥n¥r");
}
void main ( ) {
    counter = 0;
    rtos_run ( );
}


////////////////////////////////////////////////////////////////////////////////////////////////////
// This file demonstrates how to use the real time operating systems    //
// messaging functions                                                  //
//                                                                      //
// this demo makes use of the PIC18F452 prototyping board               //
////////////////////////////////////////////////////////////////////////////////////////////////////

#include <18F452.h>
#device ICD=TRUE
#fuses HS,NOLVP,NOWDT,PUT
#use delay(clock=20000000)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7)
#use rtos(timer=0,minor_cycle=100ms)
int8 count;
// each task will now be given a two byte queue
#task(rate=1000ms,max=100ms,queue=2)
void The_first_rtos_task ( );
#task(rate=500ms,max=100ms,queue=2)
void The_second_rtos_task ( );
void The_first_rtos_task ( ) {
    // the function rtos_msg_poll will return the number of messages in the
    // current tasks queue
    // always make sure to check that their is a message or else the read
    // function will hang
    if(rtos_msg_poll ( )>0){
        // the function rtos_msg_read, reads the first value in the queue
        printf("messages recieved by task1 : %i¥n¥r",rtos_msg_read ( ));
        // the funciton rtos_msg_send, sends the value given as the
        // second parameter to the function given as the first
        rtos_msg_send(The_second_rtos_task,count);
        count++;
    }
}
void The_second_rtos_task ( ) {
    rtos_msg_send(The_first_rtos_task,count);
    if(rtos_msg_poll ( )>0){
        printf("messages recieved by task2 : %i¥n¥r",rtos_msg_read ( ));
        count++;
    }
}
void main ( ) {
    count=0;
```

```
    rtos_run();
}


////////////////////////////////////////////////////////////////////////////////////////////////////
// This file demonstrates how to use the real time operating systems       //
// yield function                                                          //
//                                                                         //
// this demo makes use of the PIC18F452 prototyping board                  //
////////////////////////////////////////////////////////////////////////////////////////////////////

#include <18F452.h>
#device ICD=TRUE
#fuses HS,NOLVP,NOWDT,PUT
#use delay(clock=20000000)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7)
#use rtos(timer=0,minor_cycle=100ms)
#task(rate=1000ms,max=100ms,queue=2)
void The_first_rtos_task ( );
#task(rate=500ms,max=100ms,queue=2)
void The_second_rtos_task ( );
void The_first_rtos_task ( ) {
    int count=0;
    // rtos_yield allows the user to break out of a task at a given point
    // and return to the same ponit when the task comes back into context
    while(TRUE){
        count++;
        rtos_msg_send(The_second_rtos_task,count);
        rtos_yield ( );
    }
}
void The_second_rtos_task ( ) {
    if(rtos_msg_poll( ))
    {
        printf("count is : %i¥n¥r",rtos_msg_read ( ));
    }
}
void main ( ) {
    rtos_run();
}


////////////////////////////////////////////////////////////////////////////////////////////////////
// This file demonstrates how to use the real time operating systems       //
// signal and wait function to handle resources                            //
//                                                                         //
// this demo makes use of the PIC18F452 prototyping board                  //
////////////////////////////////////////////////////////////////////////////////////////////////////

#include <18F452.h>
#device ICD=TRUE
#fuses HS,NOLVP,NOWDT,PUT
#use delay(clock=20000000)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7)
#use rtos(timer=0,minor_cycle=100ms)
// a semaphore is simply a shared system resource
// in the case of this example, the semaphore will be the red LED
```

```
int8 sem;
#define RED PIN_B5
#task(rate=1000ms,max=100ms,queue=2)
void The_first_rtos_task ( );
#task(rate=1000ms,max=100ms,queue=2)
void The_second_rtos_task ( );
void The_first_rtos_task ( ) {
    int i;
    // this will decrement the semaphore variable to zero which signals
    // that no more user may use the resource
    rtos_wait(sem);
    for(i=0;i<5;i++){
        output_low(RED); delay_ms(20); output_high(RED);
        rtos_yield ( );
    }
    // this will inrement the semaphore variable to zero which then signals
    // that the resource is available for use
    rtos_signal(sem);
}
void The_second_rtos_task ( ) {
    int i;
    rtos_wait(sem);
    for(i=0;i<5;i++){
        output_high(RED); delay_ms(20); output_low(RED);
        rtos_yield ( );
    }
    rtos_signal(sem);
}
void main ( ) {
    // sem is initialized to the number of users allowed by the resource
    // in the case of the LED and most other resources that limit is one
    sem=1;
    rtos_run();
}


////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// This file demonstrates how to use the real time operating systems await     //
// function                                                                    //
//                                                                             //
// this demo makes use of the PIC18F452 prototyping board                      //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <18F452.h>
#device ICD=TRUE
#fuses HS,NOLVP,NOWDT,PUT
#use delay(clock=20000000)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7)
#use rtos(timer=0,minor_cycle=100ms)
#define RED PIN_B5
#define GREEN PIN_A5
int8 count;
#task(rate=1000ms,max=100ms,queue=2)
void The_first_rtos_task ( );
#task(rate=1000ms,max=100ms,queue=2)
void The_second_rtos_task ( );
void The_first_rtos_task ( ) {
```

```
    // rtos_await simply waits for the given expression to be true
    // if it is not true, it acts like an rtos_yield and passes the system
    // to the next task
    rtos_await(count==10);
    output_low(GREEN); delay_ms(20); output_high(GREEN);
    count=0;
}
void The_second_rtos_task ( ) {
    output_low(RED); delay_ms(20); output_high(RED);
    count++;
}
void main ( ) {
    count=0;
    rtos_run();
}


////////////////////////////////////////////////////////////////////////////////////////////////////
// This file demonstrates how to use the operating systems statistics       //
// features                                                                 //
//                                                                          //
// this demo makes use of the PIC18F452 prototyping board                   //
////////////////////////////////////////////////////////////////////////////////////////////////////

#include <18F452.h>
#device ICD=TRUE
#fuses HS,NOLVP,NOWDT,PUT
#use delay(clock=20000000)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7)
#use rtos(timer=0,minor_cycle=100ms,statistics)
// This structure must be defined inorder to retrieve the statistical
// information
struct rtos_stats {
    int32 task_total_ticks;          // number of ticks the task has used
    int16 task_min_ticks;             // the minimum number of ticks used
    int16 task_max_ticks;              // the maximum number of ticks ueed
    int16 hns_per_tick;                // us = (ticks*hns_per_tic)/10
};
#task(rate=1000ms,max=100ms)
void The_first_rtos_task ( );
#task(rate=1000ms,max=100ms)
void The_second_rtos_task ( );
void The_first_rtos_task ( ) {
    struct rtos_stats stats;
    rtos_stats(The_second_rtos_task,&stats);
    printf ( "¥n¥r" );
    printf ( "task_total_ticks : %Lius¥n¥r" ,
                (int32)(stats.task_total_ticks)*stats.hns_per_tick );
    printf ( "task_min_ticks     : %Lius¥n¥r" ,
                (int32)(stats.task_min_ticks)*stats.hns_per_tick );
    printf ( "task_max_ticks     : %Lius¥n¥r" ,
                (int32)(stats.task_max_ticks)*stats.hns_per_tick );
    printf ("¥n¥r");
}
void The_second_rtos_task ( ) {
    int i, count = 0;
    while(TRUE) {
```

```
        if(rtos_overrun(the_second_rtos_task)) {
            printf("The Second Task has Overrun¥n¥r¥n¥r");
            count=0;
        }
        else
           count++;
        for(i=0;i<count;i++)
            delay_ms(50);

        rtos_yield();
    }
}
void main ( ) {
    rtos_run ( );
}


////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// This file demonstrates how to create a basic command line using the serial     //
// port withought having to stop RTOS operation, this can also be considered      //
// a semi kernal for the RTOS                                                     //
//                                                                                //
// this demo makes use of the PIC18F452 prototyping board                         //
////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <18F452.h>
#device ICD=TRUE
#fuses HS,NOLVP,NOWDT,PUT
#use delay(clock=20000000)
#use rs232(baud=9600,xmit=PIN_C6,rcv=PIN_C7)
#use rtos(timer=0,minor_cycle=100ms)
#define RED PIN_B5
#define GREEN PIN_A5
#include <string.h>
// this character array will be used to take input from the prompt
char input [ 30 ];
// this will hold the current position in the array
int index;
// this will signal to the kernal that input is ready to be processed
int1 input_ready;
// different commands
char en1 [ ] = "enable1";
char en2 [ ] = "enable2";
char dis1 [ ] = "disable1";
char dis2 [ ] = "disable2";
#task(rate=1000ms,max=100ms)
void The_first_rtos_task ( );
#task(rate=1000ms,max=100ms)
void The_second_rtos_task ( );
#task(rate=500ms,max=100ms)
void The_kernal ( );
// serial interupt
#int_rda
void serial_interrupt ( )
{
    if(index<29) {
        input [ index ] = getc ( );     // get the value in the serial recieve reg
```

```
        putc ( input [ index ] );         // display it on the screen
        if(input[index]==0x0d){           // if the input was enter
            putc('¥n');
            input [ index ] = '¥0';       // add the null character
            input_ready=TRUE;             // set the input read variable to true
            index=0;                      // and reset the index
        }
        else if (input[index]==0x08){
            if ( index > 1 ) {
                putc(' ');
                putc(0x08);
                index-=2;
            }
        }
        index++;
    }
    else {
        putc ( '¥n' );
        putc ( '¥r' );
        input [ index ] = '¥0';
        index = 0;
        input_ready = TRUE;
    }
}
void The_first_rtos_task ( ) {
    output_low(RED); delay_ms(50); output_high(RED);
}
void The_second_rtos_task ( ) {
    output_low(GREEN); delay_ms(20); output_high(GREEN);
}
void The_kernal ( ) {
    while ( TRUE ) {
        printf ( "INPUT:> " );
        while(!input_ready)
            rtos_yield ( );
        printf ( "%S¥n¥r%S¥n¥r", input , en1 );
        if ( !strcmp( input , en1 ) )
            rtos_enable ( The_first_rtos_task );
        else if ( !strcmp( input , en2 ) )
            rtos_enable ( The_second_rtos_task );
        else if ( !strcmp( input , dis1 ) )
            rtos_disable ( The_first_rtos_task );
        else if ( !strcmp ( input , dis2 ) )
            rtos_disable ( The_second_rtos_task );
        else
            printf ( "Error: unknown command¥n¥r" );
        input_ready=FALSE;
        index=0;
    }
}
void main ( ) {
    // initialize input variables
    index=0;
    input_ready=FALSE;
    // initialize interrupts
    enable_interrupts(int_rda);
    enable_interrupts(global);
    rtos_run();
```

```
}
```