Scheduling / Tasking

Print PDF

Search

Search
(1) Introduction
(2) Traffic Lights
IoT WiFi
Advanced Topics
Scheduling
Plugin - Running C
How To

External Links

Creative Science Centre

- Alternatives

Scheduling was originally left out of the language for a variety of reasons, see the Alternatives for those reasons. With the advent of the use of the WiFi module it is now more relevant to be able to run something in the background and still have the 'ok' prompt. From version 2.3 serial ..71 scheduling is included.

As this is really more like multi tasking the prefix to the commands is 'task'.

Advantages:

- There is no context switching as needed in interrupts so making the process very efficient
- Small individual programs can be written making programming much easier
- Timed events much easier to implement
- Actionable (buttons) events much easier to implement
- Period is in mS, maximum value 4,294,967,295 (49 days)

Limitations:

- Priority has a value from 1 to 50
- Maximum number of task entries is 20, may be limited also by memory
- Care must be taken not to take up too much CPU time
- Task loop may not produce accurate timings
- Will only run when the ok prompt is visible ** See function taskup()
- Uses Timer 1 *

* Initially the core CPU timer was used. However using the implementation of this timer on the PIC32 it was not possible to derive a stable timer. It just kept stopping at random and so the timing is now provided by TIMER 1

Functions:

- **taskadd("text",priority,period,action)**

Adds or creates a task slot. "text" can be any valid instruction, function or command. In other words anything that can be entered to the 'ok' prompt without causing an error.

Action = 0 This is a one off event, after the event fires the task slot will be deleted.

Action = 1 This is a continuous task activity, the text will be carried out at every set period.

Action = 2 Pauses the activity, use with taskmod() to stop and start the individual task.

Example:

x = taskadd("print \"Hello\"",1,5000,1) // prints "Hello" every 5 seconds

Note that x will contain the slot number, this can be used for later modifying or deleting the task entry. If negative then that indicates an error.

Example:

Flash an LED, assume that LED(1) turns the led on and LED(0) turns it off

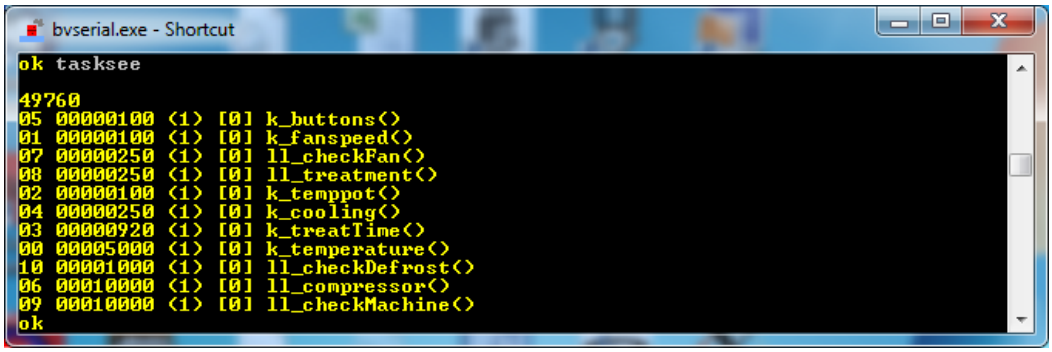
```
function do_flash()
  LED(1) // turn the led on
  taskadd("LED(0)",5,50,0) // see text below
endf

slotL1 = taskadd("do_flash()",2,5000,1)
```

In the above the 'do_flash()' function is scheduled to be called every 5 seconds with a priority of 2 and an action of 1 that means it will be called every 5 seconds continuously. When function do_flash() is called it turns on the led and a task is created to tune off the led 50mS after. There are some IMPORTANT points to note:

1. do_flash() can be called at any time, it does not need to be called by another task, calling will simply flash the led.
2. The function is non-blocking, it does not hold up the processor as it would if we used a wait() function
3. The action of 0 will mean that when the task inside do_flash() has been executed it will be deleted from the task list.
4. The led can be stopped from flashing by using taskmod(slotL1,3,2) ans started by taskmod(slotL1,3,1). This sets the action of task slotL1 to 2 (paused) or 1 (continuous)

- **tasksee()**

Displays the task entries in priority order.



[slot number] [period] [(action)] [health] Task to run

Slot number: This is the task slot and can be referred to when modifying the task, see taskmod() below. The value is also returned when creating the task with taskadd().

Period: This is the specified task period in mS

(Action): This can have 3 possible values, 0 is one off and will be removed from the list when it is fired. 1 is continuous and will therefor be running. 2 is paused.

[Health]: Ideally this should be 0. If it is not zero then it means that it has missed that number of requested periods. To explain further, if health is not 0 then it means that when the task manager added the next period to the current period it was still less than the current time and so the period is added again and health incremented until the next period is in the future.

It may not matter that the health is not 0. To reduce the health count either increase the task time or reduce the time that the function takes (make the function smaller).

- **taskmod(slot,n,x)**

Modifies an existing task. It is not possible to modify the text just the other items.

slot is the slot number as found from tasksee(). In general when using taskadd() the next available slot will be used, however if there is a deleted slot then that will be used first and so tasksee() should be used first.

n = 1 to modify the priority
n = 2 to modify the period
n = 3 to modify the action

Example

taskmod(0,2,10000) // modify slot period to 10 seconds

- **taskdel(slot)**

Deletes an individual taskedule using the slot number.

- **taskclr()**

Clears all of the taskedule entries

- **taskctl(n)**

Task control. This enables or disables the scheduler.

n = 0 // stops all task activities

n = 1 // starts all activities, the task activities will be activated if their time is up

n = 2 // re-starts all activities. The task items will be started from the beginning

- **tick()**

Returns the value of the CPU timer in mS. The timer is set to 0 on reset.

- **taskup()**

Task update. A task will only normally run when ByPic is doing nothing, i.e. waiting for user input. Whole programs can be written with just using tasks in the background and if tasks are used this is probably the best way to do it.

It may be that for some reason a loop is required for example:

```
function x()
   while 1
      .. do code
   wend
endf
```

In the above function no tasks would be executed. If tasks are also required then use taskup() within the loop, this is similar to processMessages in Windows:

```
function x()
   while 1
      taskup()
      .. do code
   wend
endf
```

Tasks will now be processed within the loop. A word of warning though, taskup() needs to save the context (as is also required for an interrupt) before running and this is a reasonable overhead each time it is called.

Timing

When a period is selected for a scheduled event, the time is checked against the processor tick and so the time is accurate. However it is only checked when taskchk() is run and as this is not called (cannot be called) in much less then 1mS intervals, then the actual period may be slightly larger than the period selected which will depend on when taskchk() is called. The period will never be smaller.

Real World Examples

The scenario is that we have a gas boiler, the start up sequence is:

1. purge the boiler which involves turning on a fan so that any residual gas is removed (10 sec)
2. start the igniter (5 sec)
3. turn on the gas
4. Turn off the igniter 5 seconds after the gas has lit

```
function start_up()
  purge_on()
  taskadd("purge_off()",1,10000,0) // one off
  taskadd("ignite_on()",1,10000,0)
  taskadd("gas_on()",1,15000,0)
  ... check we have ignition and fail if not
  taskadd("ignite_off(),1,20000,0)
endf
```

Obviously for an actual example there will be safety checks but this shows how a sequence can be formed by using a one off taskedule. The function start_up() will return immediately and so other inputs can be checked whilst this procedure is taking place - an emergency stop button for example.