805xRTOS

Generated by Doxygen 1.6.3

Fri Jun 11 17:48:33 2010

Contents

Chapter 1

RTOS

Author

Vinicius Kursancew

This is a simple RTOS optimized for the 8051 processor. Currently it supports the KeilC51 compiler. It has the following features:

- Scheduler with priority levels, using round-robin for tasks of same-priority
- Suspend/Resume tasks
- Mutexes
- Binary Semaphores
- ~2.5kbyte code footprint with keil level 7 optimizer
- RAM footprint starting from ~25 bytes iram + ~80 bytes xram, xram grows according to number of tasks and priorities
- User defined trap for erroneous behavior

Notice that avoiding priority inversion is left to the user since no priority inheritance is currently implemented.

Code size can be reduced by turning on/off features, using the following macros in the rtos.h file will disable each feature:

```
#define noSUSPEND
#define noMUTEX
#define noSEMAPHORE
#define noPREEMPTION
#define noTRAP
```

The macros STACK_START, MAX_STACK_SIZE, MAX_TASK_COUNT, MAX_-PRIORITIES, MAX_MUTEX_SHARE, MAX_NAME found in rtos.h affect the configuration of the RTOS.

VERY IMPORTANT THINGS BEFORE USING:

- DO NOT use keil optimizer above level 7 (default is 8), above level 7 it messes up the stack with it's optimizations.
- Always check if define STACK_START is configured correctly. It must match the value from the .M51 file generated by keil (look for a line similar to the one below).

```
IDATA   001FH     0001H     UNIT         ?STACK
          ^^ This is the STACK_SIZE, so use #define STACK_START 0x1f
```

- Another important setting is going to the Target Options dialog on the 'BL51 Misc' tab. On the OVERLAY textbox add:

  ```
  k_signalto ! *, k_waiton ! *, k_acquire ! *, k_release ! *,
  k_yield ! *, k_suspend ! *, k_resume ! *, schedule ! *,
  wake_task ! *, timer_isr ! *
  ```

  This is important so Keil does not share ram used by those routines (Keil cannot figure the calltree when you manipulate the stack on the rtos and it guesses it wrong). On the same list you also should add your tasks entry points using the following format: `* ! task_entry_function .` As an example, suppose you have something like:

  ```
  k_create_task(idle, "idl", 0);
  k_create_task(task1, "t1", 1);
  k_start();
  ```

  the final configuration would look (notice the invertion of $*$ in relation to ! for the tasks:

  ```
  k_signalto ! *, k_waiton ! *, k_acquire ! *, k_release ! *,
  k_yield ! *, k_suspend ! *, k_resume ! *, schedule ! *,
  wake_task ! *, timer_isr ! *, * ! idle, * ! task1
  ```

2010, Vinicius Kursancew

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Module Documentation

5.1 rtos

Data Structures

- struct task
- struct generic_sync

  structure that represents a generic sync construct (mutex or semaphore)

Modules

- Documentation of data and functions that are used only within rtos.c

Defines

- #define STACK_START 0x1d

  VERY IMPORTANT Starting value for the SP, the compiler will give you this value.

- #define MAX_STACK_SIZE 30

  Maximum stack size, the user should estimate this by checking the maximum depth that function calls may go.

- #define MAX_NAME 4

  Maximum size of the name for each task, affects ram used.

- #define MAX_TASK_COUNT 4

  Maximum number of tasks that can be created, affects ram used and may not be more than 31.

- #define MAX_PRIORITIES 3

Maximum number of priorities allowed, affects ram used.

- #define MAX_MUTEX_SHARE 3

  Maximum number of tasks that may share a mutex, affects ram used per mutex.

- #define noPREEMPTION
- #define noTRAP
- #define XRAM_t(t, decl) t xdata decl
- #define IDATA_t(t, decl) t idata decl
- #define XRAM_PTR_t(t, decl) t xdata * decl
- #define IDATA_TO_XRAM_PTR_t(t, decl) t xdata * idata decl
- #define XRAM_PTR_cast(t, var) (t xdata *)(var)
- #define PRIORITY_BITMASK_t(decl) unsigned char idata decl[MAX_-PRIORITIES]
- #define TASK_NULL 0xFF

  Special value that means NULL for a task.

- #define SETUP_TIMER TL0 = 0xb0;TH0 = 0x3c;TMOD &= 0xF0;TMOD |= 1;ET0 = 1;TR0 = 1;

  Macro to configure timer0.

- #define CBEGIN ea_save|=_testbit_(EA); ea_save <<= 1;

  Saves EA and clears, entering a critical region.

- #define CEND ea_save >>= 1; EA=(ea_save&1);

  Restores EA, exiting the critical region.

Typedefs

- typedef unsigned char byte_t
- typedef void(code * voidf)(void)

Functions

- char k_task_create (voidf fun, const char *name, byte_t prio)
- void k_start ()
- void k_yield ()
- void k_suspend (byte_t t)
- void k_resume (byte_t t)
- void k_create_semaphore (struct generic_sync xdata *sem)
- void k_waiton (struct generic_sync xdata *sem)
- void k_signalto (struct generic_sync xdata *sem)
- void k_create_mutex (struct generic_sync xdata *mut)
- void k_acquire (struct generic_sync xdata *mut)
- void k_release (struct generic_sync xdata *mut)

- void k_user_trap (const char *cause)

  User defined trap, called on internal errors.

5.1.1 Detailed Description

Author

Vinicius Kursancew

5.1.2 Define Documentation

5.1.2.1 #define CBEGIN ea_save|=_testbit_(EA); ea_save <<= 1;

Saves EA and clears, entering a critical region.

Definition at line 138 of file rtos.h.

5.1.2.2 #define CEND ea_save >>= 1; EA=(ea_save&1);

Restores EA, exiting the critical region.

Definition at line 140 of file rtos.h.

5.1.2.3 #define IDATA_t(t, decl) t idata decl

Definition at line 110 of file rtos.h.

5.1.2.4 #define IDATA_TO_XRAM_PTR_t(t, decl) t xdata * idata decl

Definition at line 112 of file rtos.h.

5.1.2.5 #define MAX_MUTEX_SHARE 3

Maximum number of tasks that may share a mutex, affects ram used per mutex.

Definition at line 100 of file rtos.h.

5.1.2.6 #define MAX_NAME 4

Maximum size of the name for each task, affects ram used.

Definition at line 91 of file rtos.h.

5.1.2.7 #define MAX_PRIORITIES 3

Maximum number of priorities allowed, affects ram used.

Definition at line 97 of file rtos.h.

5.1.2.8 #define MAX_STACK_SIZE 30

Maximum stack size, the user should estimate this by checking the maximum depth that function calls may go.

Definition at line 88 of file rtos.h.

5.1.2.9 #define MAX_TASK_COUNT 4

Maximum number of tasks that can be created, affects ram used and may not be more than 31.

Definition at line 94 of file rtos.h.

5.1.2.10 #define noPREEMPTION

Definition at line 106 of file rtos.h.

5.1.2.11 #define noTRAP

Definition at line 107 of file rtos.h.

5.1.2.12 #define PRIORITY_BITMASK_t(decl) unsigned char idata decl[MAX_PRIORITIES]

Definition at line 121 of file rtos.h.

5.1.2.13 #define SETUP_TIMER TL0 = 0xb0;TH0 = 0x3c;TMOD &= 0xF0;TMOD |= 1;ET0 = 1;TR0 = 1;

Macro to configure timer0.

Definition at line 128 of file rtos.h.

5.1.2.14 #define STACK_START 0x1d

VERY IMPORTANT Starting value for the SP, the compiler will give you this value.

If set incorrectly the kernel will crash when doing context switch.

Definition at line 78 of file rtos.h.

5.1.2.15 #define TASK_NULL 0xFF

Special value that means NULL for a task.

Definition at line 125 of file rtos.h.

5.1.2.16 #define XRAM_PTR_cast(t, var) (t xdata *)(var)

Definition at line 113 of file rtos.h.

5.1.2.17 #define XRAM_PTR_t(t, decl) t xdata * decl

Definition at line 111 of file rtos.h.

5.1.2.18 #define XRAM_t(t, decl) t xdata decl

Definition at line 109 of file rtos.h.

5.1.3 Typedef Documentation

5.1.3.1 typedef unsigned char byte_t

Definition at line 132 of file rtos.h.

5.1.3.2 typedef void(code * voidf)(void)

Defined as a function pointer to a function of prototype void fun(void);

Definition at line 153 of file rtos.h.

5.1.4 Function Documentation

5.1.4.1 void k_acquire (struct generic_sync xdata * *mut*)

Tryes to acquire a mutex. If the mutex is already acquired, blocks until it is released.

Parameters

mut pointer to mutex to try to acquire

5.1.4.2 void k_create_mutex (struct generic_sync xdata * *mut*)

Initializes a mutex object.

Parameters

mut pointer to generic_sync structure that will represent the mutex

5.1.4.3 void k_create_semaphore (struct generic_sync xdata * *sem*)

Initializes a semaphore. It is initially unavailable.

Parameters

sem pointer to a generic_sync structure that will hold the semaphore

5.1.4.4 void k_release (struct generic_sync xdata * *mut*)

Releases an acquired mutex and wake tasks pending on it. Notice that releasing an un-acquired mutex, or a mutex acquired by another task is illegal.

Parameters

mut pointer to mutex to be release

5.1.4.5 void k_resume (byte_t *t*)

Resumes execution of a task, making it ready to run iff its not BLOCKED

Parameters

t task to resume

5.1.4.6 void k_signalto (struct generic_sync xdata * *sem*)

Signals a semaphore and wakes the task sleeping on it (if any).

Parameters

sem pointer to semaphore to signal

5.1.4.7 void k_start ()

Starts the scheduler and never returns. It is very important to notice that the user must provide the IDLE task. The idle task is a task that is always ready to execute, a simple definition would be:

```
void idle()
{
        for(;;) k_yield();
}
```

and then, before calling k_start():

```
k_create_task(idle, "idl", 0);
```

5.1.4.8 void k_suspend (byte_t *t*)

Suspends current task, it will only be ready again if k_resume is called on the task

Parameters

t task number that should be suspended, passing TASK_NULL will suspend currently running task

5.1.4.9 char k_task_create (voidf *fun*, const char * *name*, byte_t *prio*)

Creates a task. The task is marked as ready to run.

Parameters

fun function with prototype void fun(void) when control is passed to the task

name a string that holds a name that represents the task

prio priority of the task, 0 being the lowest, macro MAX_PRIORITIES defines the maximum

Returns

task number

5.1.4.10 void k_user_trap (const char * *cause*)

User defined trap, called on internal errors.

5.1.4.11 void k_waiton (struct generic_sync xdata * *sem*)

Waits on a semaphopre until it becomes available. Returns immediately if it's already available.

Parameters

sem pointer to semaphore to waiton

5.1.4.12 void k_yield ()

Causes a context switch.

5.2 Documentation of data and functions that are used only within rtos.c

Data Structures

- struct task

Defines

- #define READY 0
- #define BLOCKED 1
- #define SUSPENDED 2
- #define SUSPENDED_BLOCKED 3
- #define SAVE
- #define RESTORE
- #define TO_XRAM
- #define TO_STACK
- #define TASK_SET_READY_MASK(index) ready_masks[tasks[(index)].prio] |= 1<<((index));
- #define TASK_CLR_READY_MASK(index) ready_masks[tasks[(index)].prio] &= ~(1<<((index)));

Functions

- PRIORITY_BITMASK_t (ready_masks)
- PRIORITY_BITMASK_t (rr_masks)
- void push_bank ()
- void pop_bank ()
- void returni ()
- static void schedule ()
- static void wake_task (byte_t t)
- void timer_isr ()

Variables

- struct task xdata tasks [MAX_TASK_COUNT]
- struct task xdata *idata current_task
- byte_t idata current_task_index = 0
- byte_t idata task_count = 0
- volatile byte_t idata ea_save = 0
- byte_t idata ea_save

5.2.1 Define Documentation

5.2.1.1 #define BLOCKED 1

Definition at line 19 of file rtos.c.

5.2.1.2 #define READY 0

Definition at line 17 of file rtos.c.

5.2.1.3 #define RESTORE

Value:

```
_pop_(PSW);     \
        pop_bank();     \
        _pop_(DPH);     \
        _pop_(DPL);     \
        _pop_(B);       \
        _pop_(IE);      \
        _pop_(ACC);
```

Restores context.

Definition at line 108 of file rtos.c.

5.2.1.4 #define SAVE

Value:

```
_push_(ACC);    \
        _push_(IE);             \
        _push_(B);              \
        _push_(DPL);    \
        _push_(DPH);    \
        push_bank();    \
        _push_(PSW);
```

Save context to stack.

Definition at line 96 of file rtos.c.

5.2.1.5 #define SUSPENDED 2

Definition at line 21 of file rtos.c.

5.2.1.6 #define SUSPENDED_BLOCKED 3

Definition at line 23 of file rtos.c.

5.2.1.7 #define TASK_CLR_READY_MASK(index) ready_masks[tasks[(index)].prio] &= ~(1<<((index)));

Macro used to mark a task as not ready. Parameter is the task index from the tasks[] array After this the task cannod be scheduled to run until TASK_SET_READY_MASK is called again on it

Definition at line 147 of file rtos.c.

5.2.1.8 #define TASK_SET_READY_MASK(index) ready_masks[tasks[(index)].prio] |= 1<<((index));

Macro used to mark a task as ready, parameter is the task index from the tasks[] array

Definition at line 141 of file rtos.c.

5.2.1.9 #define TO_STACK

Value:

```
{ \
        byte_t idata * idata ram = (byte_t idata *)STACK_START; \
        IDATA_TO_XRAM_PTR_t(byte_t,  xram) = current_task->stack_copy;  \
        SP = STACK_START + current_task->sp; \
        while((byte_t)ram <= SP) *(ram++) = *(xram++); }
```

Writes bytes from the current task saved context back to the stack region in the IDATA ram.

Definition at line 131 of file rtos.c.

5.2.1.10 #define TO_XRAM

Value:

```
{ \
        byte_t idata * idata ram = (byte_t idata *)STACK_START; \
        IDATA_TO_XRAM_PTR_t(byte_t,  xram) = current_task->stack_copy;  \
        current_task->sp = SP - STACK_START; \
        while((byte_t)ram <= SP) *(xram++) = *(ram++); }
```

Writes bytes in the range STACK_START -> SP to the external ram into the current task stack context.

Definition at line 121 of file rtos.c.

5.2.2 Function Documentation

5.2.2.1 void pop_bank ()

Defined in low_level.a51, assembly routine to restore R0-R7 from the stack

5.2.2.2 PRIORITY_BITMASK_t (rr_masks)

Holds info of which task on each priority level was run the last time.

5.2.2.3 PRIORITY_BITMASK_t (ready_masks)

Holds which tasks are ready in an array of bitmasks, each bit representing a task

5.2.2.4 void push_bank ()

Defined in low_level.a51, assembly routine to save R0-R7 on the stack

5.2.2.5 void returni ()

Defined in low_level.a51, assembly routine to allow calling RETI from C code. This is a hack for Keil C51 because defining a function as interrupt on Keil compiler unavoidably inserts a preamble on the ISR which manipulates the stack, but in this case (conext-switch) the OS is manipulating the stack.

5.2.2.6 static void schedule () `[static]`

Finds out which is the next task to run. This is an internal function for the kernel.

Definition at line 153 of file rtos.c.

5.2.2.7 void timer_isr ()

Interrupt for timer0, in case the system is using preemption

Definition at line 296 of file rtos.c.

5.2.2.8 static void wake_task (byte_t *t*) `[static]`

Wakes up a task that is blocked putting it in the correct state. Also properly calling TASK_SET_READY_MASK. This is an internal function for the kernel.

Parameters

t task to wake up

Definition at line 184 of file rtos.c.

5.2.3 Variable Documentation

5.2.3.1 struct task xdata* idata current_task

Pointer to currently running task. It is stored on the internal ram so access is faster.

Definition at line 57 of file rtos.c.

5.2.3.2 byte_t idata current_task_index = 0

Index of the currently running task on the tasks[] array

Definition at line 62 of file rtos.c.

5.2.3.3 byte_t idata ea_save

Byte used for saving and restoring EA on calling CBEGIN and CEND macros

Definition at line 72 of file rtos.c.

5.2.3.4 volatile byte_t idata ea_save = 0

Byte used for saving and restoring EA on calling CBEGIN and CEND macros

Definition at line 72 of file rtos.c.

5.2.3.5 byte_t idata task_count = 0

Count of tasks created by calling k_task_create.

Definition at line 67 of file rtos.c.

5.2.3.6 struct task xdata tasks[MAX_TASK_COUNT]

Definition of the tasks structure. The index of this array is the corresponding's task id.

Definition at line 52 of file rtos.c.

Chapter 6

Data Structure Documentation

6.1 generic_sync Struct Reference

structure that represents a generic sync construct (mutex or semaphore)

`#include <rtos.h>`

Data Fields

- byte_t blocked

  Whether the structure is blocked or not.

- union {
  byte_t array [MAX_MUTEX_SHARE]
  Used for mutexes.
  byte_t single
  Used for semaphores.
  } wake_me

  Holds task(s) to wake.

6.1.1 Detailed Description

structure that represents a generic sync construct (mutex or semaphore)

Definition at line 143 of file rtos.h.

6.1.2 Field Documentation

6.1.2.1 byte_t generic_sync::array[MAX_MUTEX_SHARE]

Used for mutexes.

Definition at line 146 of file rtos.h.

6.1.2.2 byte_t generic_sync::blocked

Whether the structure is blocked or not.

Definition at line 144 of file rtos.h.

6.1.2.3 byte_t generic_sync::single

Used for semaphores.

Definition at line 147 of file rtos.h.

6.1.2.4 union { ... } generic_sync::wake_me

Holds task(s) to wake.

The documentation for this struct was generated from the following file:

- rtos.h

6.2 task Struct Reference

Data Fields

- byte_t stack_copy [MAX_STACK_SIZE]

  Copy of the stack for the task.

- char name [MAX_NAME+1]

  A name representing the task.

- byte_t prio

  Priority, 0=lowest, highest is defined on the MAX_PRIORITIES macro.

- byte_t state

  Current state of the task, may be: READY, BLOCKED, SUSPENDED or SUSPENDED_BLOCKED.

- byte_t sp

  Count-1 of bytes that were on the stack when context switched (SP-STACK_START).

6.2.1 Detailed Description

Structure that holds information about a task. It is statically defined on the array 'tasks' and cannot hold more tasks than defined on MAX_TASK_COUNT macro.

Definition at line 40 of file rtos.c.

6.2.2 Field Documentation

6.2.2.1 char task::name[MAX_NAME+1]

A name representing the task.

Definition at line 42 of file rtos.c.

6.2.2.2 byte_t task::prio

Priority, 0=lowest, highest is defined on the MAX_PRIORITIES macro.

Definition at line 43 of file rtos.c.

6.2.2.3 byte_t task::sp

Count-1 of bytes that were on the stack when context switched (SP-STACK_START).

Definition at line 45 of file rtos.c.

6.2.2.4 byte_t task::stack_copy[MAX_STACK_SIZE]

Copy of the stack for the task.

Definition at line 41 of file rtos.c.

6.2.2.5 byte_t task::state

Current state of the task, may be: READY, BLOCKED, SUSPENDED or SUSPENDED_BLOCKED.

Definition at line 44 of file rtos.c.

The documentation for this struct was generated from the following file:

- rtos.c

Chapter 7

File Documentation

7.1 rtos.c File Reference

```
#include "rtos.h"
#include <reg51.h>
#include <intrins.h>
#include <string.h>
```

Data Structures

- struct task

Defines

- #define _s(s) #s
- #define READY 0
- #define BLOCKED 1
- #define SUSPENDED 2
- #define SUSPENDED_BLOCKED 3
- #define SAVE
- #define RESTORE
- #define TO_XRAM
- #define TO_STACK
- #define TASK_SET_READY_MASK(index) ready_masks[tasks[(index)].prio] |= 1<<((index));
- #define TASK_CLR_READY_MASK(index) ready_masks[tasks[(index)].prio] &= ~(1<<((index)));

Functions

- PRIORITY_BITMASK_t (ready_masks)
- PRIORITY_BITMASK_t (rr_masks)
- void push_bank ()
- void pop_bank ()
- void returni ()
- static void schedule ()
- static void wake_task (byte_t t)
- char k_task_create (voidf fun, const char *name, byte_t prio)
- void k_start ()
- void k_yield ()
- void timer_isr ()
- void k_suspend (byte_t t)
- void k_resume (byte_t t)
- void k_create_semaphore (struct generic_sync xdata *sem)
- void k_waiton (struct generic_sync xdata *sem)
- void k_signalto (struct generic_sync xdata *sem)
- void k_create_mutex (struct generic_sync xdata *mut)
- void k_acquire (struct generic_sync xdata *mut)
- void k_release (struct generic_sync xdata *mut)

Variables

- struct task xdata tasks [MAX_TASK_COUNT]
- struct task xdata *idata current_task
- byte_t idata current_task_index = 0
- byte_t idata task_count = 0
- volatile byte_t idata ea_save = 0

7.1.1 Define Documentation

7.1.1.1 #define _s(s) #s

Definition at line 6 of file rtos.c.

7.2 rtos.h File Reference

```
#include <reg51.h>
```

Data Structures

- struct generic_sync

  structure that represents a generic sync construct (mutex or semaphore)

Defines

- #define STACK_START 0x1d

  VERY IMPORTANT Starting value for the SP, the compiler will give you this value.

- #define MAX_STACK_SIZE 30

  Maximum stack size, the user should estimate this by checking the maximum depth that function calls may go.

- #define MAX_NAME 4

  Maximum size of the name for each task, affects ram used.

- #define MAX_TASK_COUNT 4

  Maximum number of tasks that can be created, affects ram used and may not be more than 31.

- #define MAX_PRIORITIES 3

  Maximum number of priorities allowed, affects ram used.

- #define MAX_MUTEX_SHARE 3

  Maximum number of tasks that may share a mutex, affects ram used per mutex.

- #define noPREEMPTION
- #define noTRAP
- #define XRAM_t(t, decl) t xdata decl
- #define IDATA_t(t, decl) t idata decl
- #define XRAM_PTR_t(t, decl) t xdata ∗ decl
- #define IDATA_TO_XRAM_PTR_t(t, decl) t xdata ∗ idata decl
- #define XRAM_PTR_cast(t, var) (t xdata ∗)(var)
- #define PRIORITY_BITMASK_t(decl) unsigned char idata decl[MAX_-PRIORITIES]
- #define TASK_NULL 0xFF

  Special value that means NULL for a task.

- #define SETUP_TIMER TL0 = 0xb0;TH0 = 0x3c;TMOD &= 0xF0;TMOD |= 1;ET0 = 1;TR0 = 1;

Macro to configure timer0.

- #define CBEGIN ea_save|=_testbit_(EA); ea_save <<= 1;

  Saves EA and clears, entering a critical region.

- #define CEND ea_save >>= 1; EA=(ea_save&1);

  Restores EA, exiting the critical region.

Typedefs

- typedef unsigned char byte_t
- typedef void(code * voidf)(void)

Functions

- void k_start ()
- char k_task_create (voidf fun, const char *name, byte_t prio)
- void k_yield ()
- void k_suspend (byte_t t)
- void k_resume (byte_t t)
- void k_create_semaphore (struct generic_sync xdata *sem)
- void k_waiton (struct generic_sync xdata *sem)
- void k_signalto (struct generic_sync xdata *sem)
- void k_create_mutex (struct generic_sync xdata *mut)
- void k_acquire (struct generic_sync xdata *mut)
- void k_release (struct generic_sync xdata *mut)
- void k_user_trap (const char *cause)

  User defined trap, called on internal errors.

Variables

- byte_t idata ea_save

Index