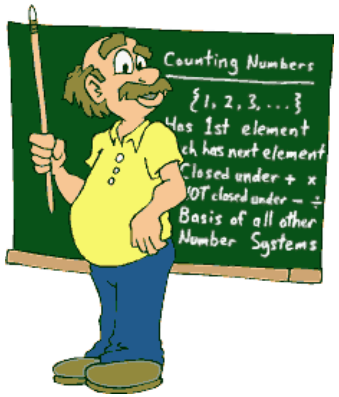
iha.dk

AMS

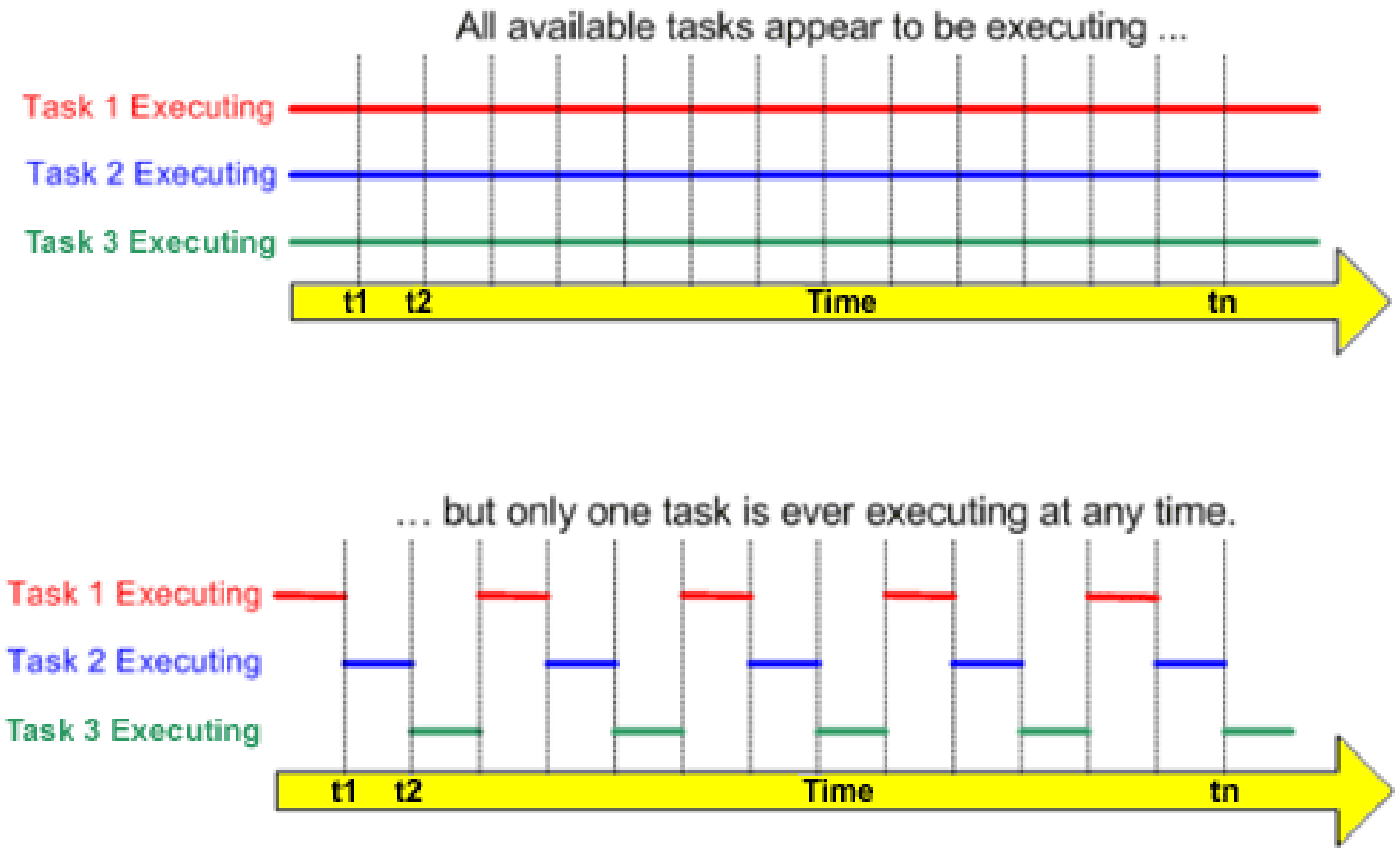
Applied Microcontroller Systems

Lesson 5: Free RTOS

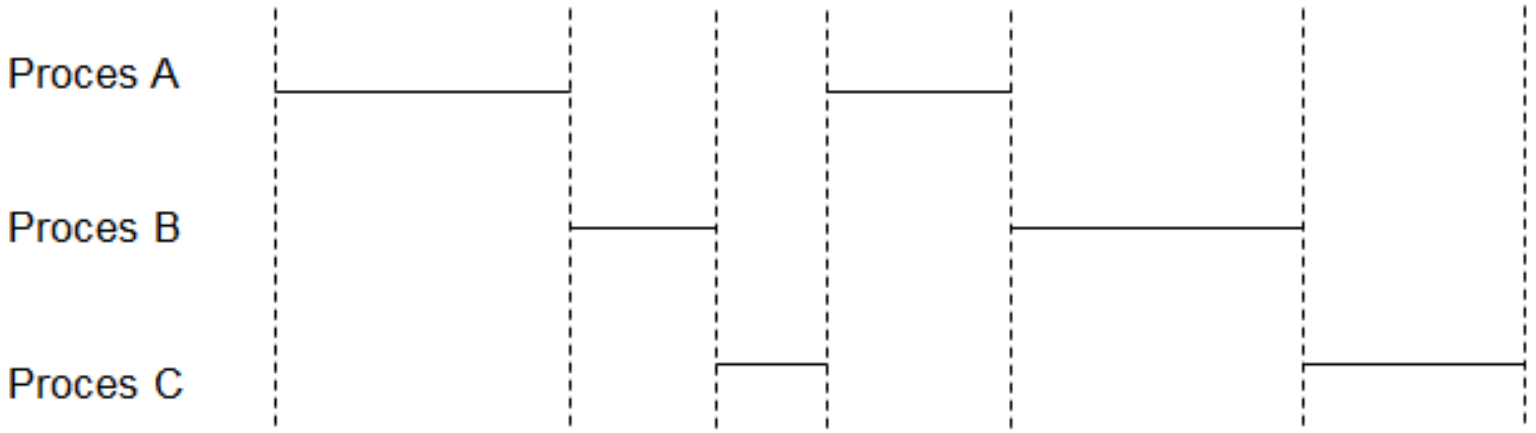


Version: February 2013, Asbjørn Baagø

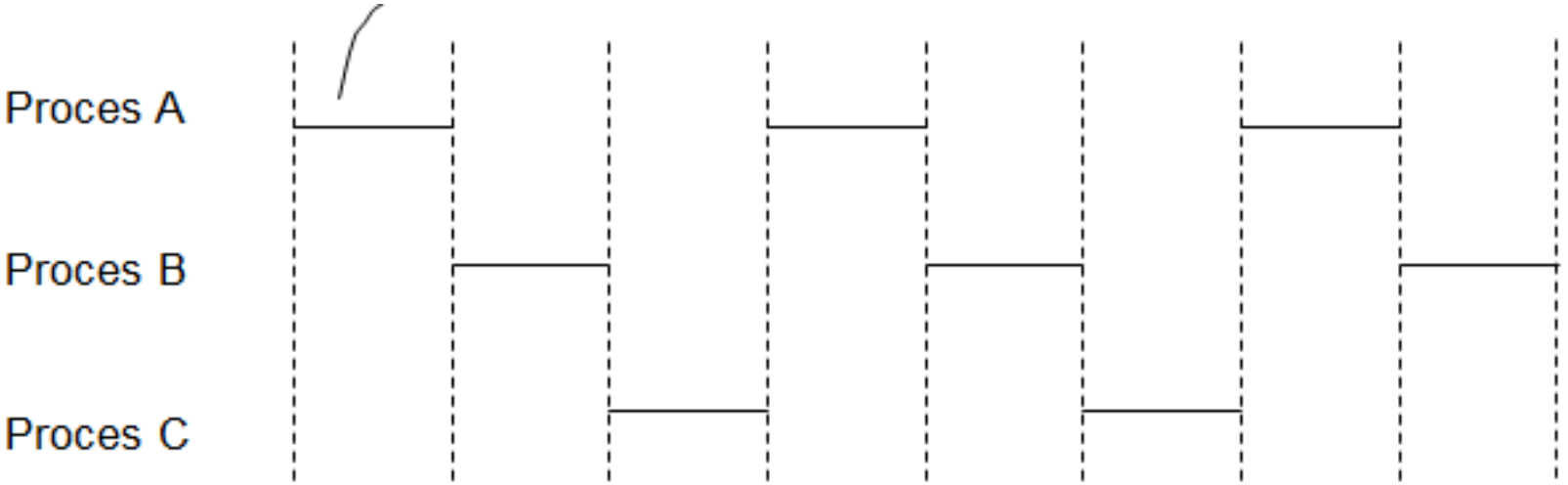
Multi-tasking



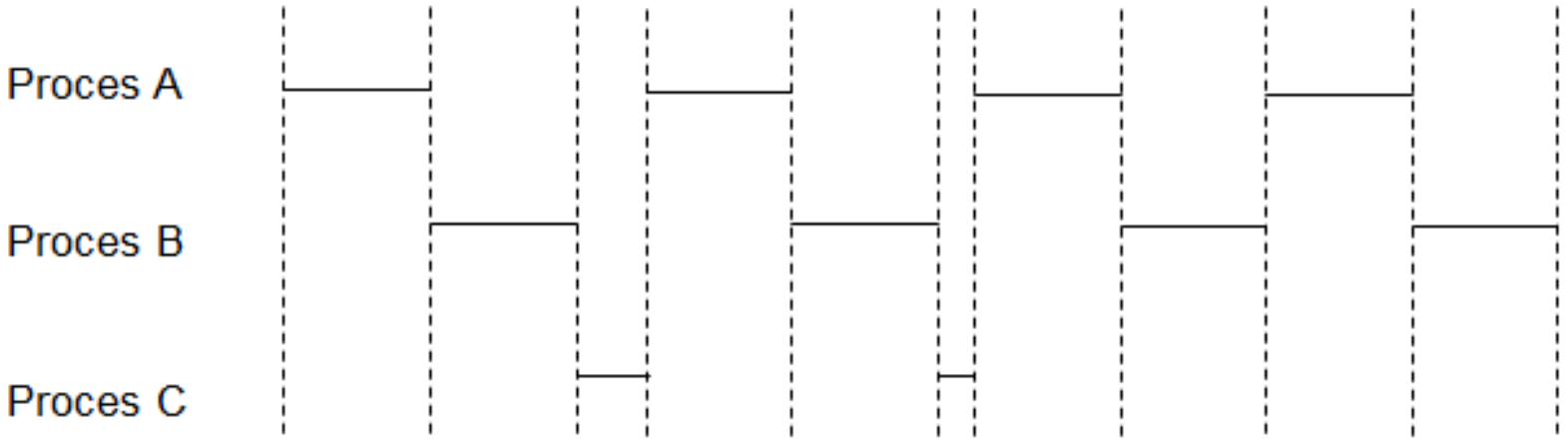
Co-operative scheduling



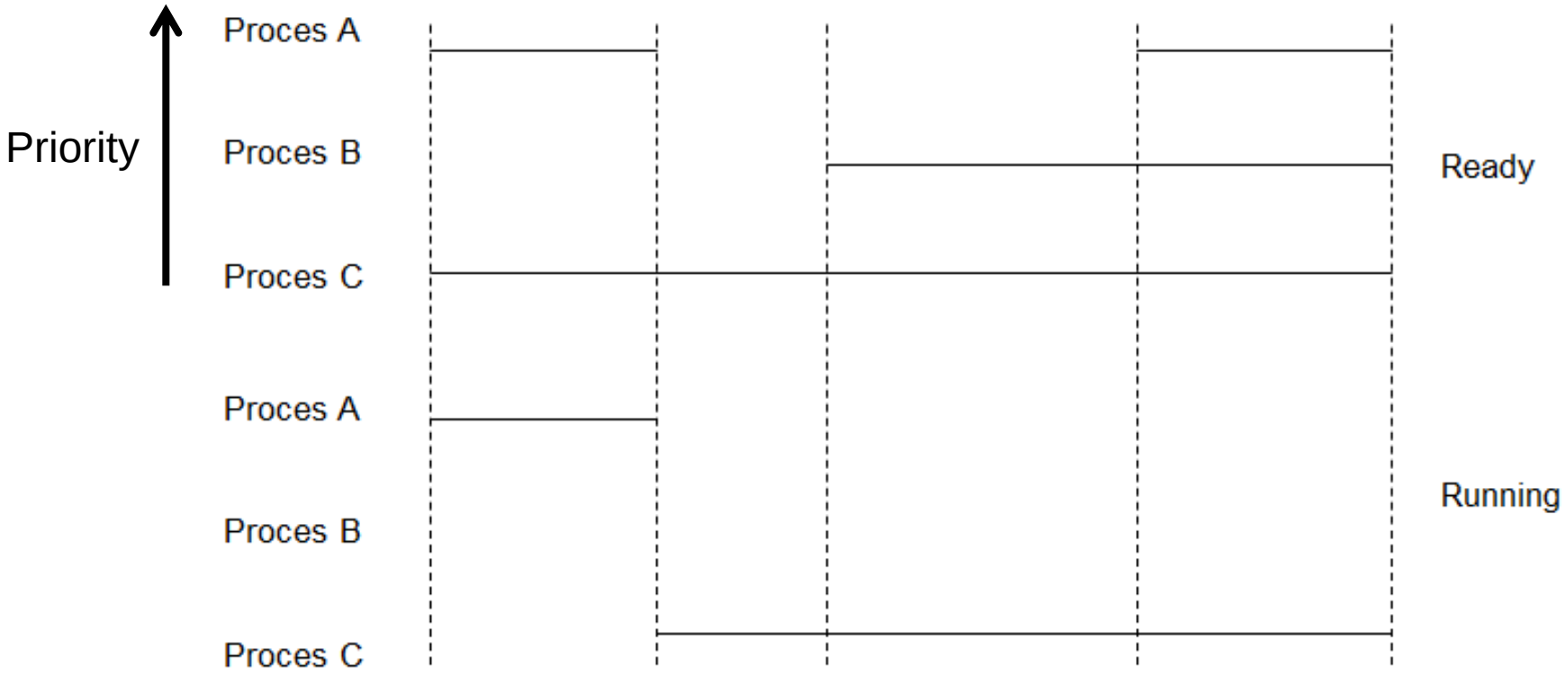
Simple time slicing



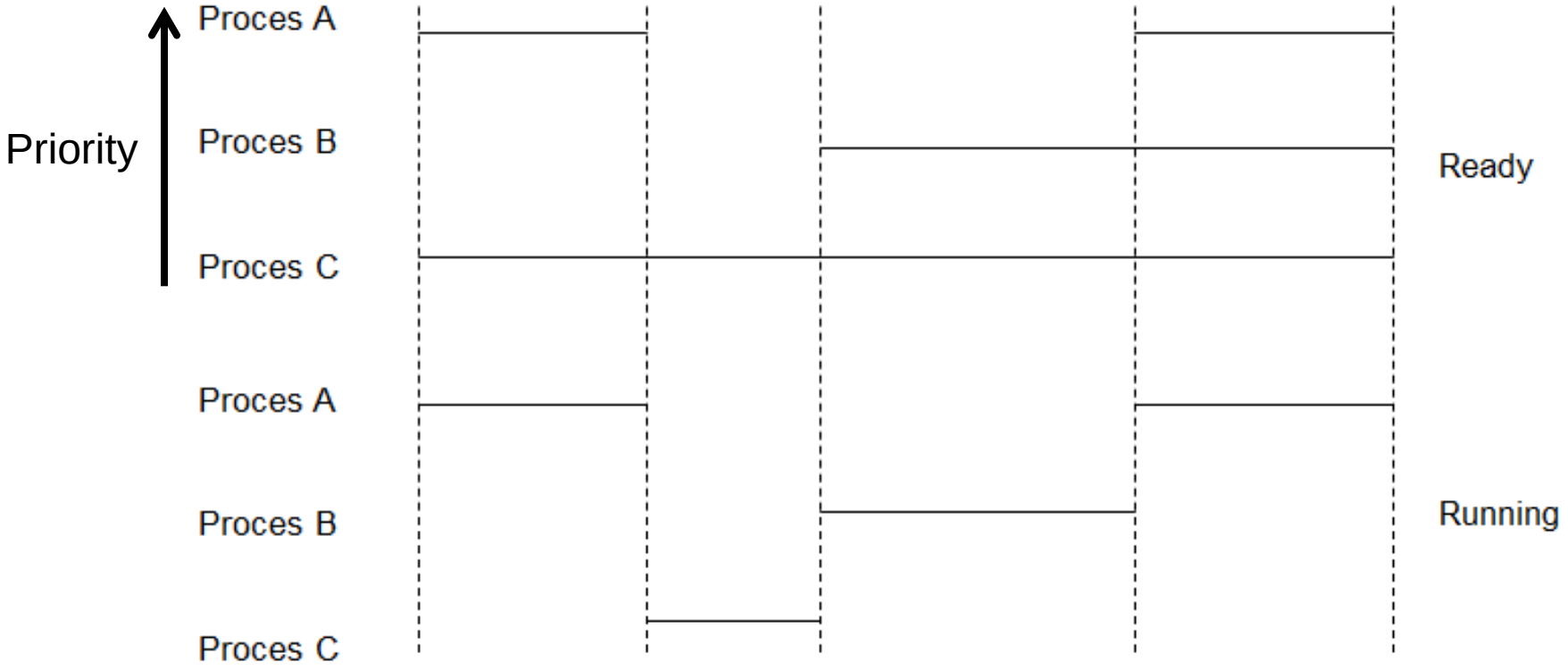
Co-operative with time slicing



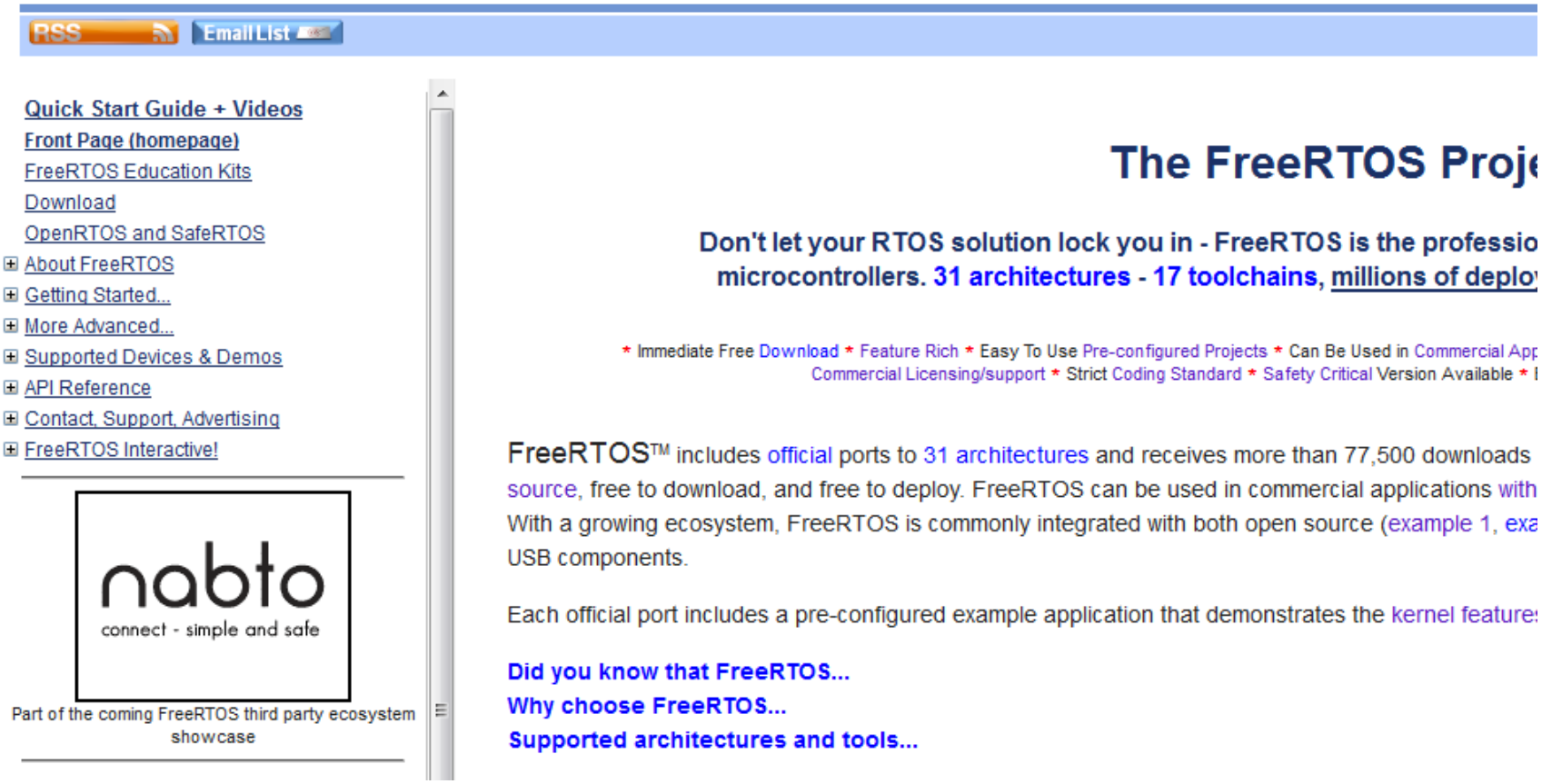
Non-preemptive priority based



Preemptive priority based



www.freertos.org



FreeRTOS Features

- Choice of RTOS scheduling policy
  - Pre-emptive:
    Always runs the highest available task.
    Tasks of identical priority share CPU time (fully pre-emptive with round robin time slicing).
  - Cooperative:
    Context switches only occur if a task blocks, or explicitly calls taskYIELD().
- Message queues
- Semaphores [via macros]
- Mutexes

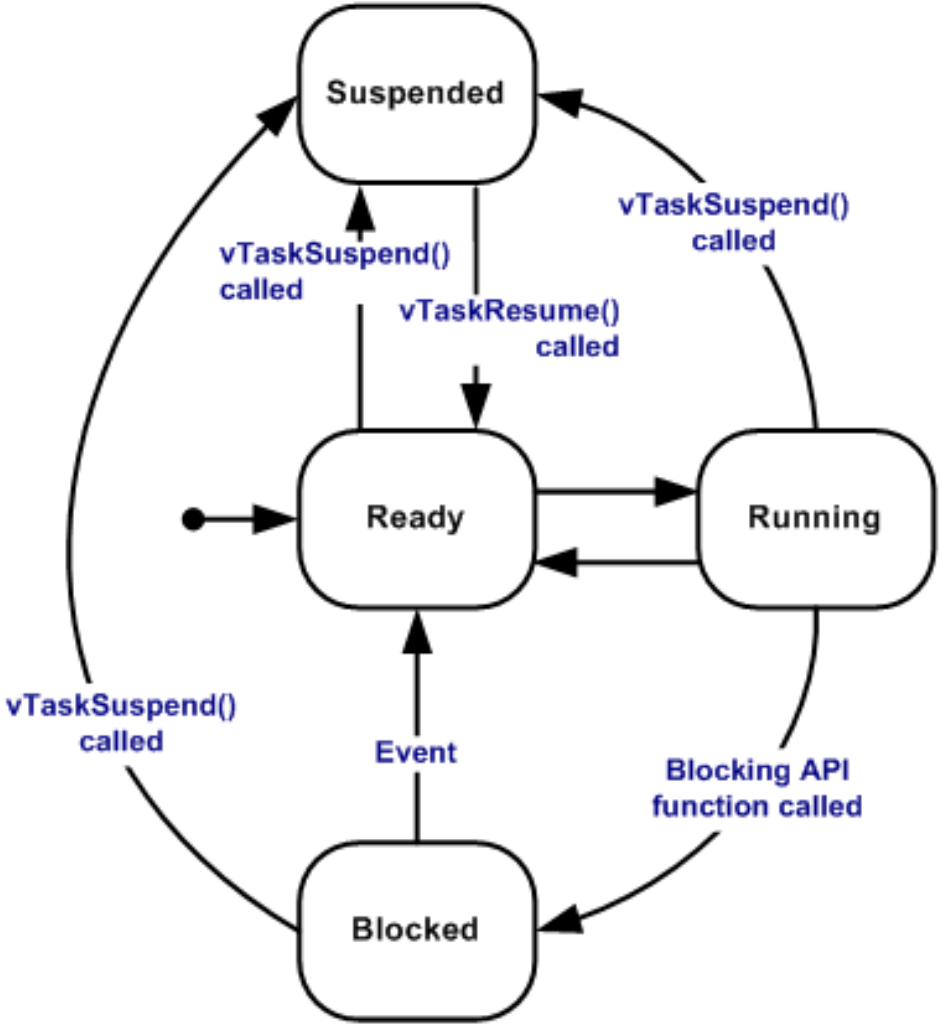
Design Philosophy

- Nearly all the code is written in C, with only a few assembler functions where completely unavoidable
- any number of tasks can share the same priority
- Simple, Portable, Concise

Task states

- **Running**
  When a task is actually executing it is said to be in the Running state. It is currently utilizing the processor.
- **Ready**
  Ready tasks are those that are able to execute (they are not blocked or suspended) but are not currently executing because a different task (of equal or higher priority, if using pre-emptive) is already in the Running state.
- **Blocked**
  A task is said to be in the Blocked state if it is currently waiting for either a temporal or external event. For example, if a task calls vTaskDelay() it will block (be placed into the Blocked state) until the delay period has expired - a temporal event. Tasks can also block waiting for queue and semaphore events. Tasks in the Blocked state always have a 'timeout' period, after which the task will be unblocked. Blocked tasks are not available for scheduling.
- **Suspended**
  Tasks in the Suspended state are also not available for scheduling. Tasks will only enter or exit the suspended state when explicitly commanded to do so through the vTaskSuspend() and xTaskResume() API calls respectively.
  A 'timeout' period cannot be specified.

FreeRTOS Task states



Task Creation

- xTaskCreate
  - Create a new task and add it to the list of tasks that are ready to run
- vTaskDelete
  - Remove a task from the RTOS real time kernels management. The task being deleted will be removed from all ready, blocked, suspended and event lists

Task Control

- **vTaskDelay** - Delay a task for a given number of ticks
- **vTaskDelayUntil** - Delay a task until a specified time
- **ucTaskPriorityGet** - Obtain the priority of any task
- **vTaskPrioritySet** - Set the priority of any task
- **vTaskSuspend** - Suspend any task
- **vTaskResume** - Resumes a suspended task

Kernel Control

- **vTaskStartScheduler** - Starts the real time kernel tick processing. After calling, the kernel has control over which tasks are executed and when.

- **vTaskEndScheduler** - Stops the real time kernel tick. All created tasks will be automatically deleted and multitasking (either preemptive or cooperative) will stop.

Kernel Control

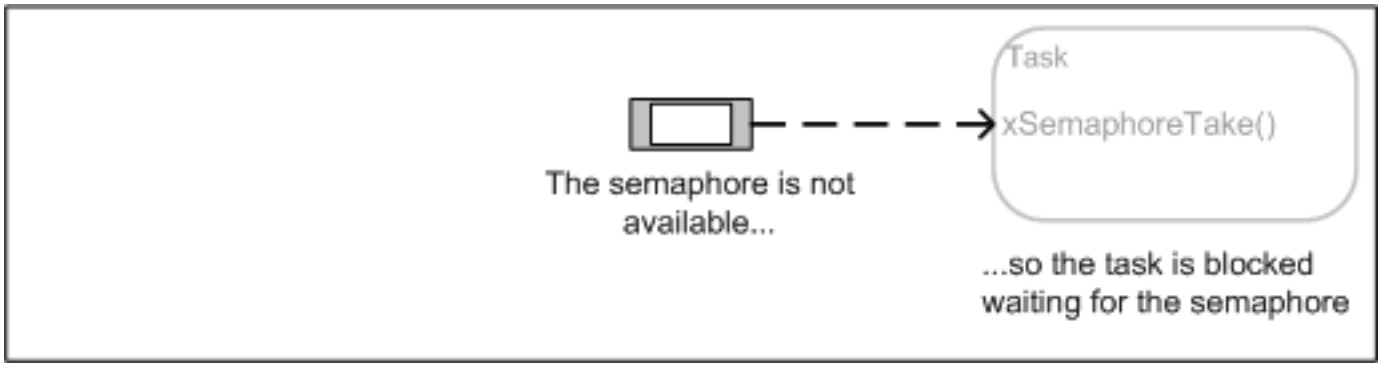
- **vTaskSuspendAll** - Suspends all real time kernel activity while keeping interrupts (including the kernel tick) enabled

- **xTaskResumeAll** - Resumes real time kernel activity following a call to vTaskSuspendAll()

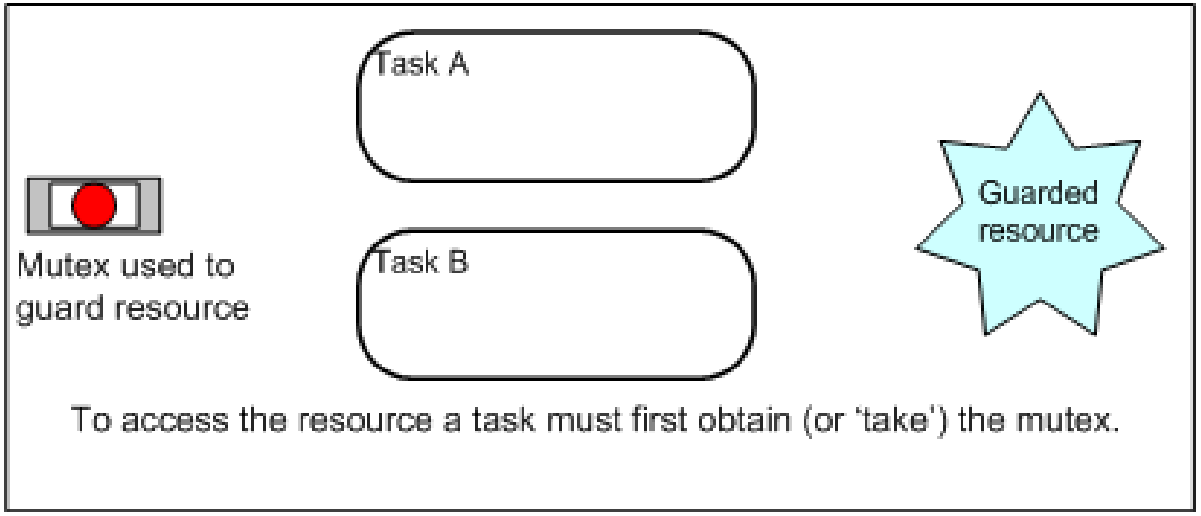
Task Utilities

- **xTaskGetTickCount** - count of ticks since vTaskStartScheduler was called
- **xTaskGetNumberOfTasks** - number of tasks that the real time kernel is currently managing
- **vTaskList** - Lists all the current tasks

Binary semaphores



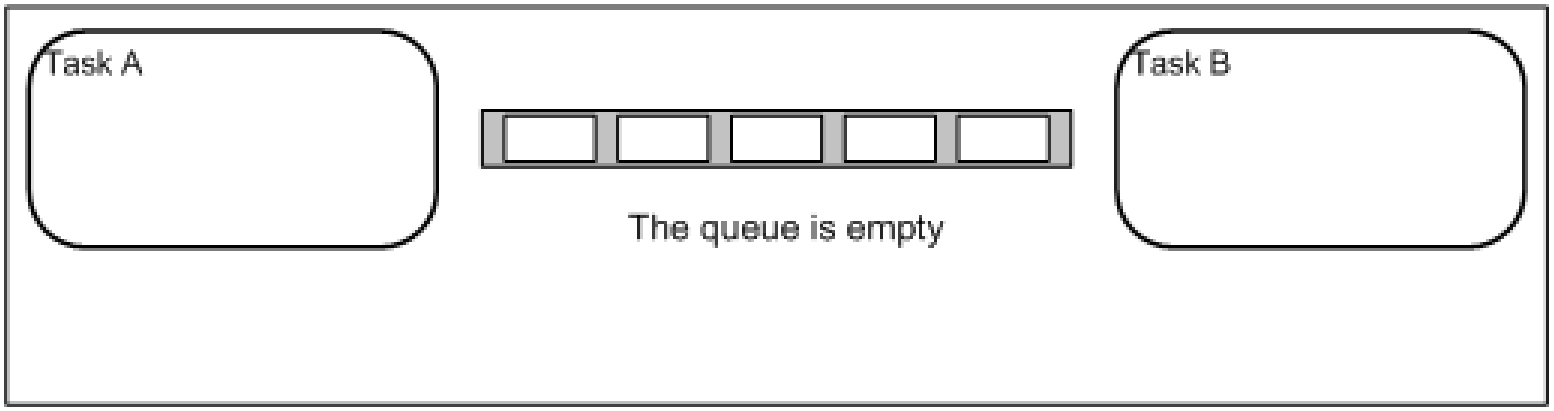
Mutex



Semaphores

- **vSemaphoreCreateBinary** – no priority inheritance, count equals only 1 and 0 values
- **xSemaphoreCreateMutex** – uses priority inheritance
- **xSemaphoreCreateCounting** - no priority inheritance, count set at create!
- **xSemaphoreTake**
- **xSemaphoreGive**
- **xSemaphoreGiveFromISR**

Queues



Queue Management

- **xQueueCreate** - Creates a new queue instance
- **xQueueSendToFront** - Post an item on a queue
- **xQueueSendToBack** - Post an item on a queue
- **xQueueReceive** - Receive an item from a queue
- **xQueuePeek** - Receive an item from a queue without removing the item form the queue
- **xQueueSendToFrontFromISR** - Post an item on a queue(safe from iSR)
- **xQueueSendToBackFromISR** - Post an item on a queue(safe from iSR)
- **xQueueReceiveFromISR** - Receive an item from a queue (safe from ISR)

Customization

Controlled by: **FreeRTOSConfig.h** file

- portUSE_PREEMPTION 1
- portCPU_CLOCK_HZ ((unsigned portLONG) 3686400)
- portTICK_RATE_HZ ((portTickType) 1000)
- portMAX_PRIORITIES ((unsigned portCHAR) 4)

Customization

- configCPU_CLOCK_HZ
  - Enter the frequency in Hz at which the internal processor core will be executing. This value is required in order to correctly configure timer peripherals
- configTICK_RATE_HZ
  - frequency of the RTOS tick interrupt
  - tick interrupt is used to measure time
  - a higher tick frequency means time can be measured to a higher resolution

NOTICE: Will use the Mega32 Timer 1 !
(Can’t be used by the application)

Customization

- ad configTICK_RATE_HZ
  - high tick frequency -> kernel will use more CPU time so be less efficient
  - More than one task can share the same priority. The kernel will share processor time between tasks of the same priority by switching between the tasks during each RTOS tick.
  - A high tick rate frequency will therefore also have the effect of reducing the 'time slice' given to each task.

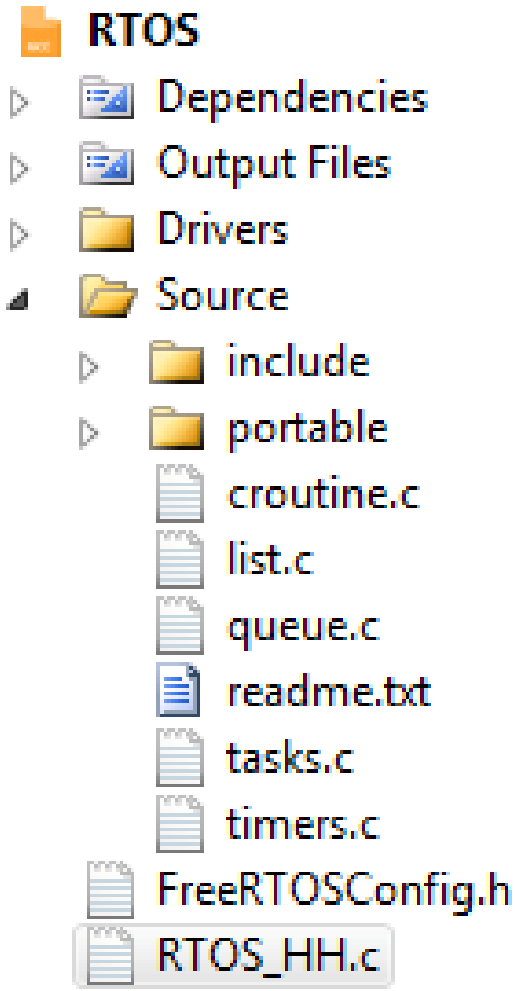
Customization

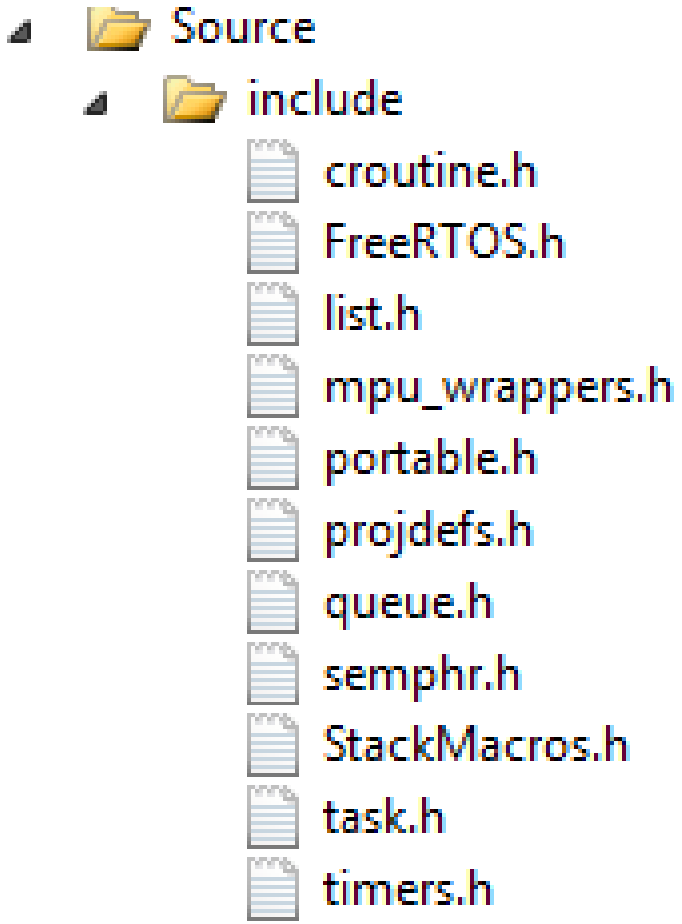
- configMAX_PRIORITIES
  - The number of priorities available to the application
  - Any number of tasks can share the same priority
  - Each available priority consumes RAM within the kernel so this value should not be set any higher than actually required by your application

Include Parameters

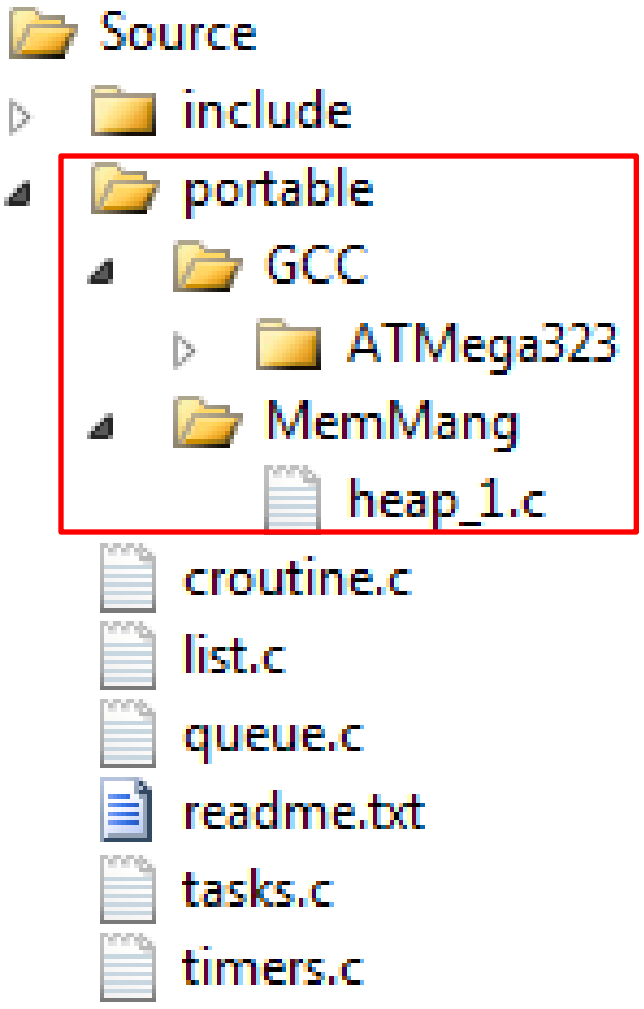
- The macros starting 'INCLUDE' allow those components of the real time kernel not utilized by application to be excluded from build
- Ensures the RTOS does not use any more ROM or RAM than necessary
- #define INCLUDE_vTaskDelete 1 (change to 0 to exclude)

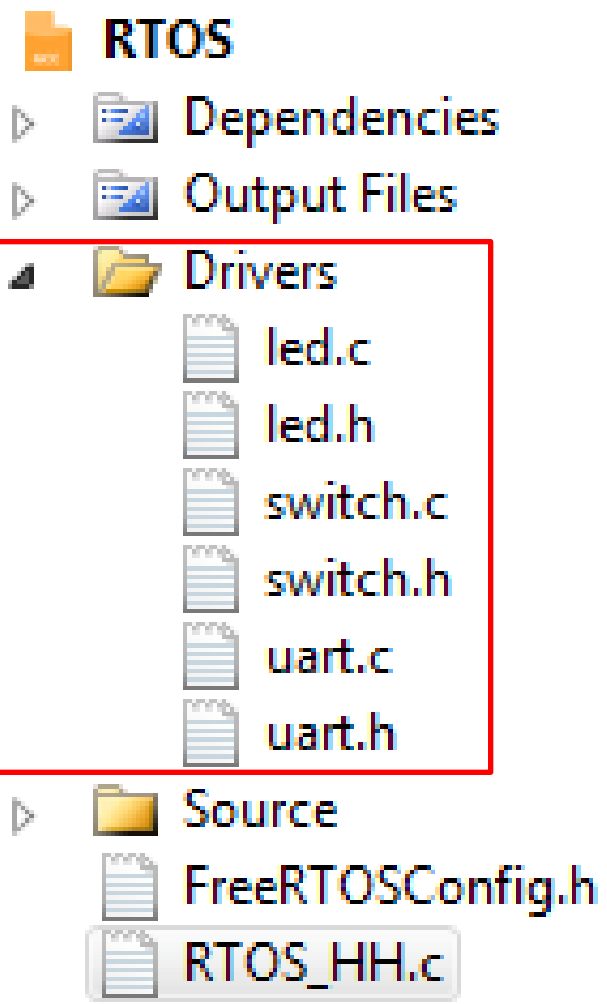
FreeRTOS in Atmel Studio 6 (Mega32)



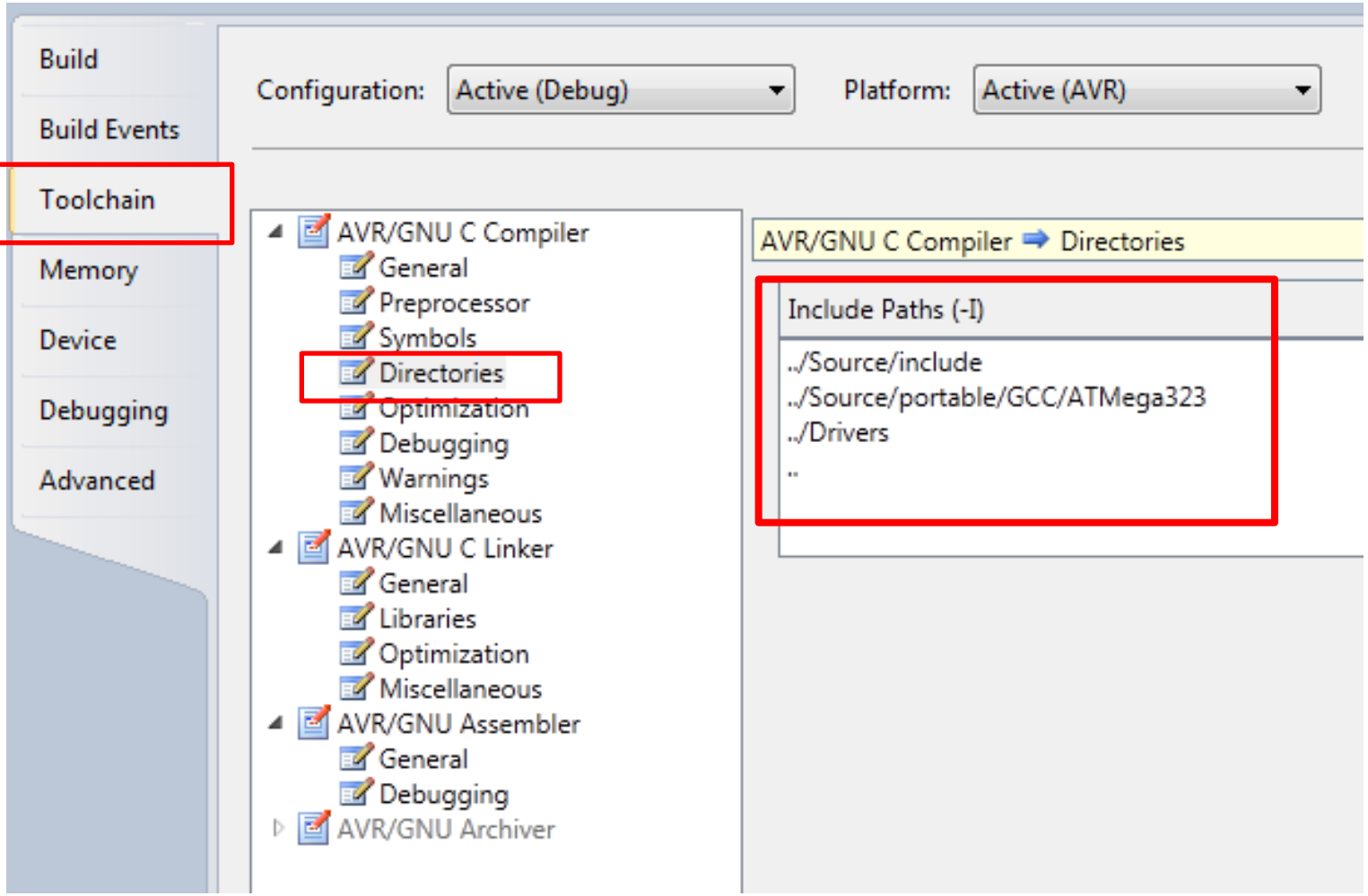


FreeRTOS in Atmel Studio 6 (Mega32)





Settings for ”Directories”



FreeRTOSConfig.h

```
#define configUSE_PREEMPTION        1
#define configUSE_IDLE_HOOK         0
#define configUSE_TICK_HOOK         0
#define configCPU_CLOCK_HZ          ( ( unsigned long ) 3686400 )
#define configTICK_RATE_HZ          ( ( portTickType ) 1000 )
#define configMAX_PRIORITIES        ( ( unsigned portBASE_TYPE ) 1 )
#define configMINIMAL_STACK_SIZE    ( ( unsigned short ) 85 )
#define configTOTAL_HEAP_SIZE       ( (size_t ) ( 3500 ) )
#define configMAX_TASK_NAME_LEN     ( 8 )
#define configUSE_TRACE_FACILITY    0
#define configUSE_16_BIT_TICKS      1
#define configIDLE_SHOULD_YIELD     1
#define configQUEUE_REGISTRY_SIZE   0

/* Co-routine definitions. */
#define configUSE_CO_ROUTINES       0
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

/* Set the following definitions to 1 to include the API function, or zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet        0
#define INCLUDE_uxTaskPriorityGet       0
#define INCLUDE_vTaskDelete             0
#define INCLUDE_vTaskCleanUpResources   0
#define INCLUDE_vTaskSuspend            0
#define INCLUDE_vTaskDelayUntil         1
#define INCLUDE_vTaskDelay              1
```

Free RTOS demo

```
/*************************************************************
FreeRTOS demo program.
Implementing 2 tasks, each blinking a LED.

STK500 setup:
  * PORTC connected to LEDS.

Henning Hargaard 8.2.2012
*************************************************************/
#include <avr/io.h>
#include "FreeRTOS.h"
#include "task.h"
#include "led.h"
```

The two tasks

```
void vLEDFlashTask1( void *pvParameters )
{
portTickType xLastWakeTime;
xLastWakeTime=xTaskGetTickCount();
  while(1)
  {
    toggleLED(0);
    vTaskDelayUntil(&xLastWakeTime,1000);
  }
}

void vLEDFlashTask2( void *pvParameters )
{
portTickType xLastWakeTime;
xLastWakeTime=xTaskGetTickCount();
  while(1)
  {
    toggleLED(1);
    vTaskDelayUntil(&xLastWakeTime,500);
  }
}
```

The main() function

```
int main(void)
{
  initLEDport();
  xTaskCreate( vLEDFlashTask1, ( signed char * ) "LED1", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );
  xTaskCreate( vLEDFlashTask2, ( signed char * ) "LED2", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL );
  vTaskStartScheduler();
  while(1)
  {}
}
```

More information

- http://www.freertos.org/
- http://www.freertos.org/a00106.html - API

End of lesson 5