FreeRTOS

Gary J. Minden
October 19, 2017

FreeRTOS

- A real-time kernel for hard real-time scheduling
  - “Hard real-time” -- Task must execute at a specific time and complete within a specific period
- Motivation
  - Abstract timing
  - Maintainability/Extensibility
  - Modularity
  - Team development
  - Easier testing
  - Code re-use
  - Improved efficiency (no polling)
  - Flexible interrupt processing
  - Mixed task types (periodic, continuous, event-driven)
  - Peripheral control (peripheral monitor)

FreeRTOS References

- Richard Barry, *Using the FreeRTOS Real-Time Kernel, ARM Cortex-M Edition*, Self-published.
  - Copyrighted and cannot be distributed on website
- http://www.freertos.org/RTOS.html (Left hand side of page)
  - Getting Started Guides
  - Examples
  - API Reference

FreeRTOS Features

- Pre-emptive or co-operative operation
- Flexible task priority assignment
- Queues
- Binary, counting, and recursive semaphores
- Mutexes
- Hook functions (Tick, Idle, Trace)
- Stack overflow checking
- Interrupt nesting

FreeRTOS Resources

- SysTick, PendingSV, SVC
- Small memory footprint -- ~ 6 KB flash, ~100 B SRAM
- Each Task
  - Flash for program code
  - SRAM for variables, stack, and heap

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

Example main.c -- Includes

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"

#include  <stddef.h>
#include  <stdbool.h>
#include  <stdint.h>
#include  <stdarg.h>

#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include “driverlib/gpio.h"

#include “Drivers/rit128x96x4.h"

#include  “Drivers/UARTStdio_Initialization.h"
#include "drivers/uartstdio.h"

#include "FreeRTOS.h"
#include "task.h"

#include "stdio.h"
```

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

Example main.c -- Time Variables

```
extern void Task_Blinky( void *pvParameters );

//*****************************************************************************
//
// The speed of the processor clock in Hertz, which is therefore the speed of the
// clock that is fed to the peripherals.
//
//*****************************************************************************
extern uint32_t          g_ulSystemClock;
```

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

Example main.c -- Initialize Processor

```
int32_t main( void ) {

    //
    // Set the clocking to run directly from the crystal.
    //
    SysCtlClockSet( SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
                    SYSCTL_XTAL_8MHZ );
      //
      // Get the system clock speed.
      //
      g_ulSystemClock = SysCtlClockGet();

      UARTStdio_Initialization();

      //
      //  Create a task to blink LED
      //
      xTaskCreate( Task_Blinky, "Blinky", 128, NULL, 1, NULL );

      UARTprintf( "FreeRTOS Starting!\n" );

      //
      //  Start FreeRTOS Task Scheduler
      //
      vTaskStartScheduler();

      while( true ) {};
}
```

Example main.c -- Set up tasks

```
//
//  Create a task to blink LED
//
xTaskCreate( Task_Blinky, "Blinky", 128, NULL, 1, NULL );
```

FreeRTOS Task Create

Task Subroutine

Task Name

Stack Size (Words)

Parameters

Priority

Task Handle

Information and Telecommunication Technology Center

KU THE UNIVERSITY OF KANSAS

Example Task_Blinky.c -- Initialize

```
/*—Task_Blinky.c
 *
 *  Author:         Gary J. Minden
 *  Organization:   KU/EECS/EECS 388
 *  Date:           2016—09—26 (B60926)
 *
 *  Description:    Blinks Status LED on Stellaris LM3S1968
 *                  Evaluation board.
 *
 */

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"

#include <stddef.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdarg.h>

#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"

#include  "FreeRTOS.h"
#include  "task.h"
```

Example Task_Blinky.c -- Initialize

```
extern void Task_Blinky( void *pvParameters ) {

      uint32_t LED_Data = 0;

          //
          // Enable the GPIO Port G.
          //
          SysCtlPeripheralEnable( SYSCTL_PERIPH_GPIOG );

          //
          // Configure GPIO_G to drive the Status LED.
          //
          GPIOPinTypeGPIOOutput( GPIO_PORTG_BASE, GPIO_PIN_2 );
          GPIOPadConfigSet( GPIO_PORTG_BASE,
                          GPIO_PIN_2, GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD );
```

Information and Telecommunication Technology Center

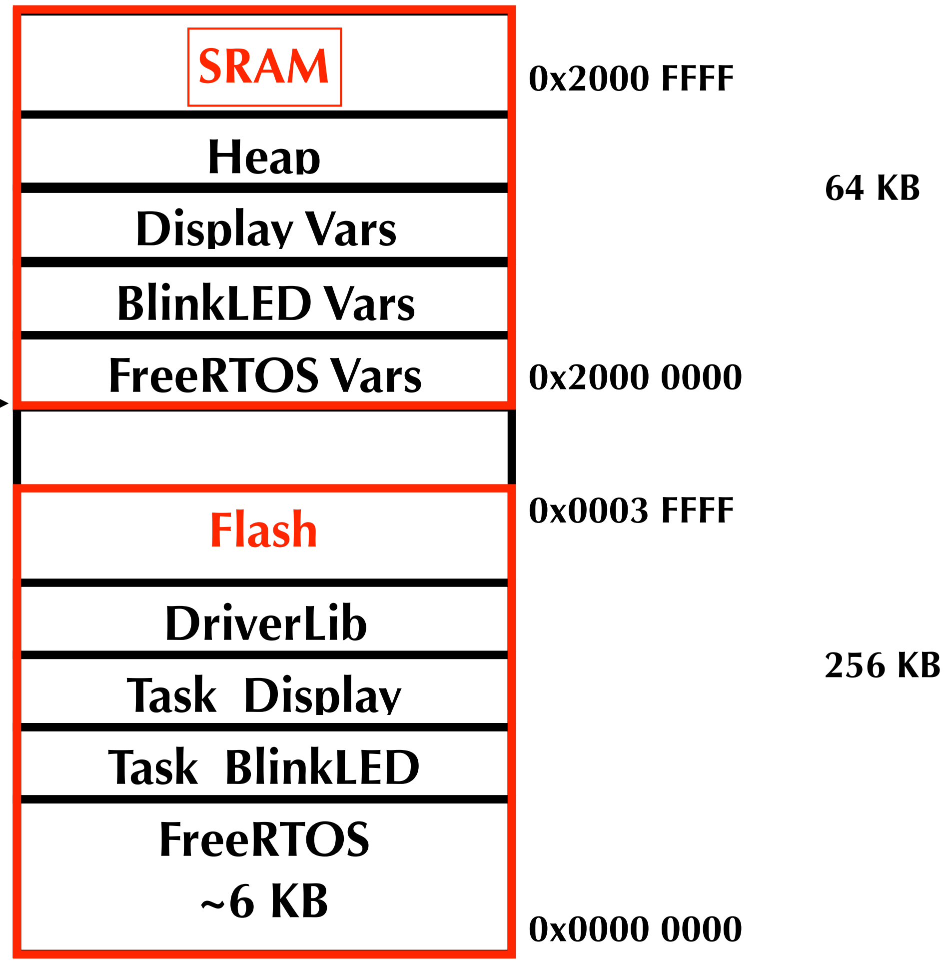
THE UNIVERSITY OF KANSAS

Example Task_Blinky.c -- Execute

```
//*****************************************************************************
//
//    Task execution.
//
//*****************************************************************************

      while ( true ) {
        //
        // Toggle the LED.
        //
         LED_Data = GPIOPinRead( GPIO_PORTG_BASE, GPIO_PIN_2 );
         LED_Data = LED_Data ^ 0x04;
         GPIOPinWrite( GPIO_PORTG_BASE, GPIO_PIN_2, LED_Data );

        //
        //   Delay 50 mS.
        //
         vTaskDelay( pdMS_TO_TICKS( 50 ) );
      }
}
```

FreeRTOSMemory Map

- Variables
- Stack

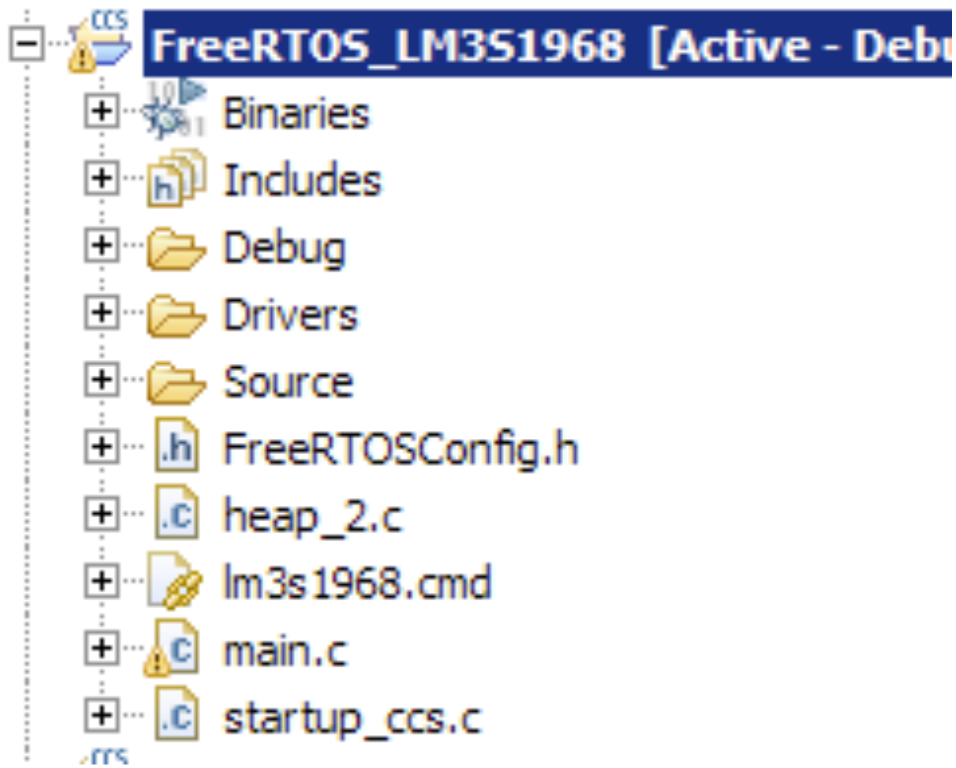


- Programs
- Constants
- Interrupt Vectors

FreeRTOS Organization

- Queues and Lists -- queue.c and list.c
- Timers -- timers.c
- Task Management -- tasks.c
- Porting -- port.c, portmacro.h, portasm.s
- Configuration -- FreeRTOSConfig.h
- Application -- main.c, Task_BlinkLED.c Task_Display.c, DriverLib

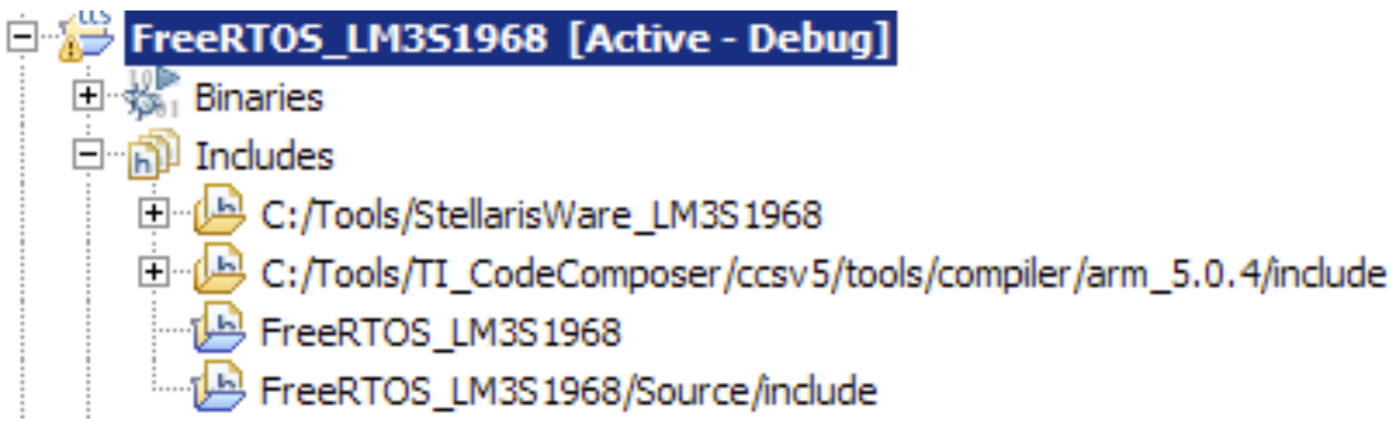
FreeRTOS Project Structure



Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

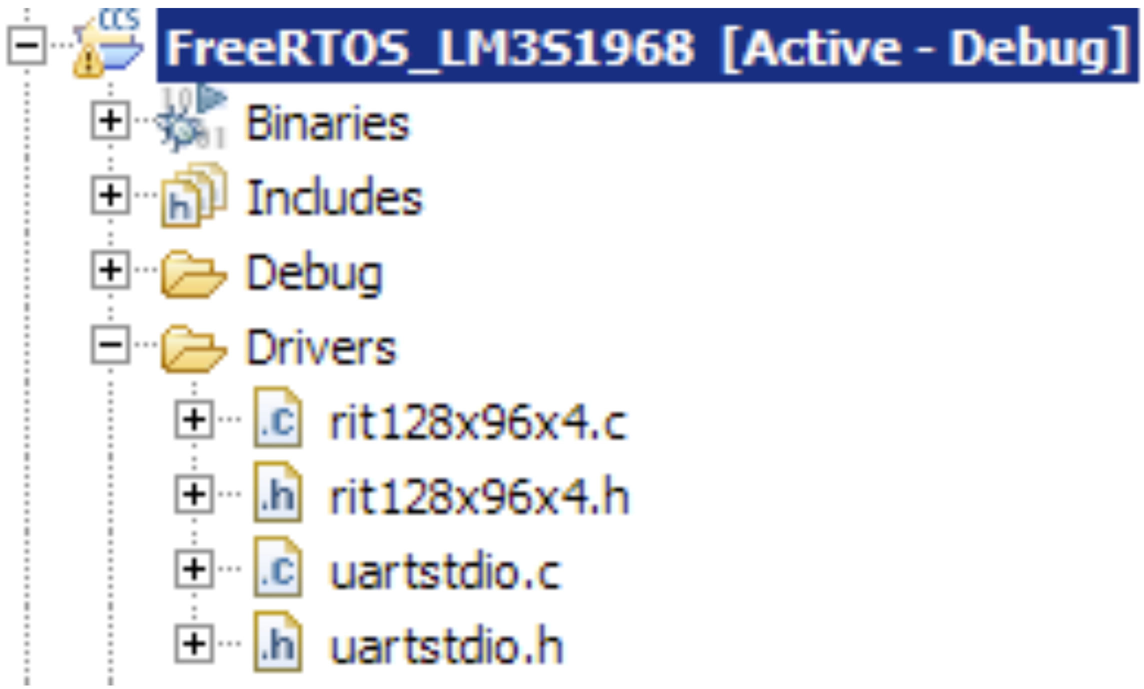
FreeRTOS Include Files



Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

FreeRTOS Driver Modules



Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

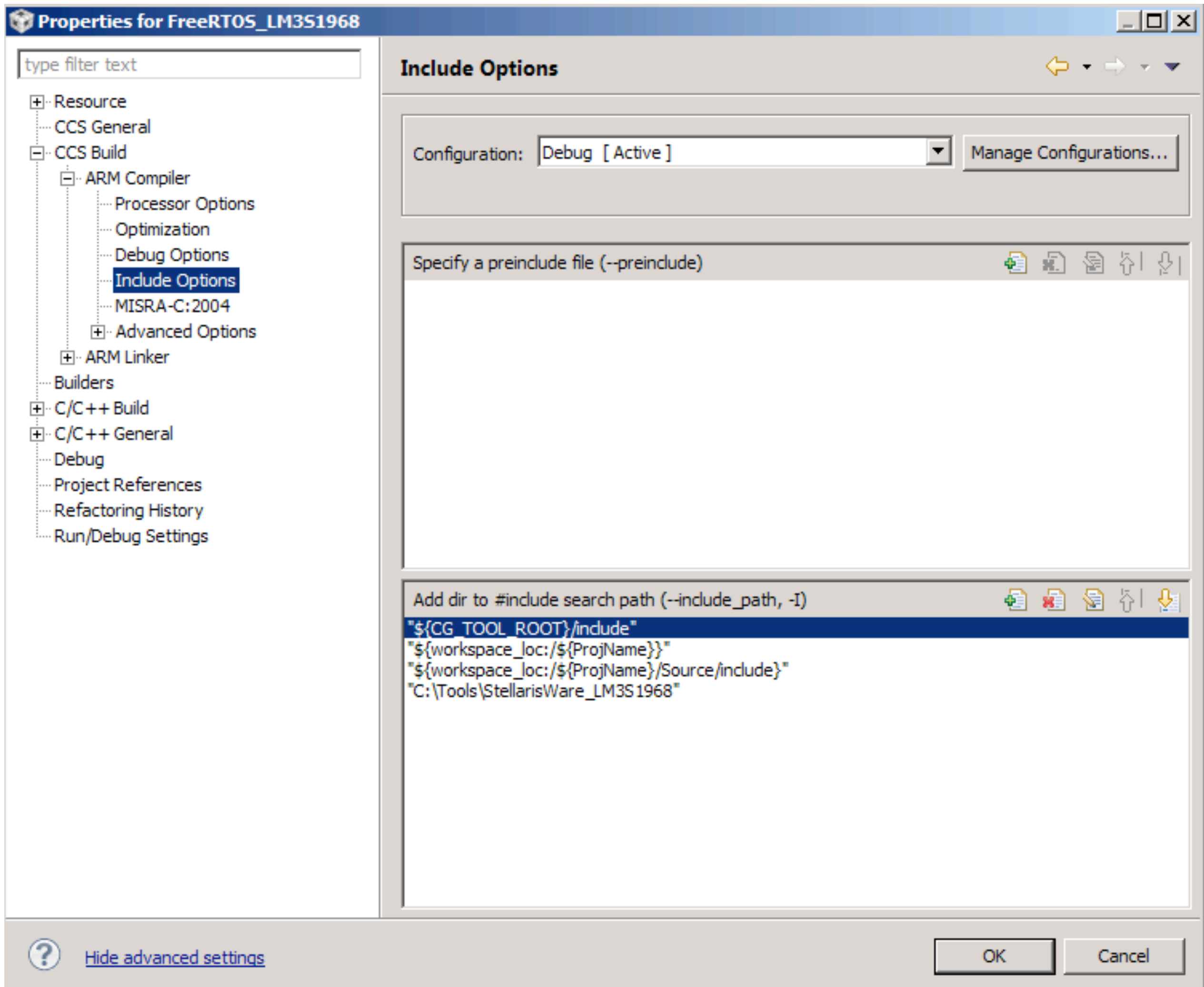
FreeRTOS Program Modules

- Source
  - include
  - portable
    - CCS5
      - ARM_CM3
        - port.c
        - portmacro.h
        - portasm.s
    - readme.txt
  - croutine.c
  - list.c
  - queue.c
  - tasks.c
  - timers.c
  - readme.txt

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

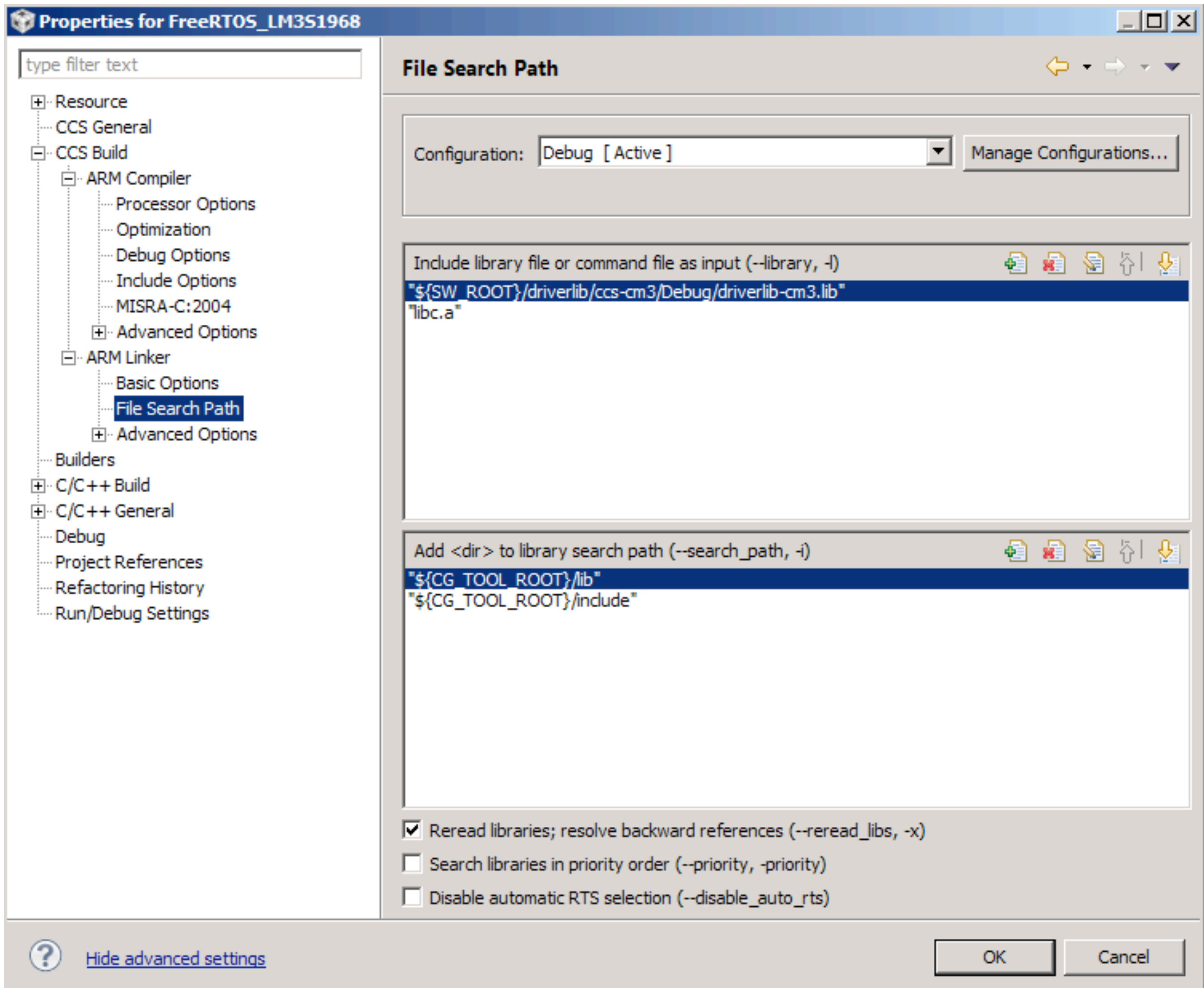
Code Composer -- Setting Includes



Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

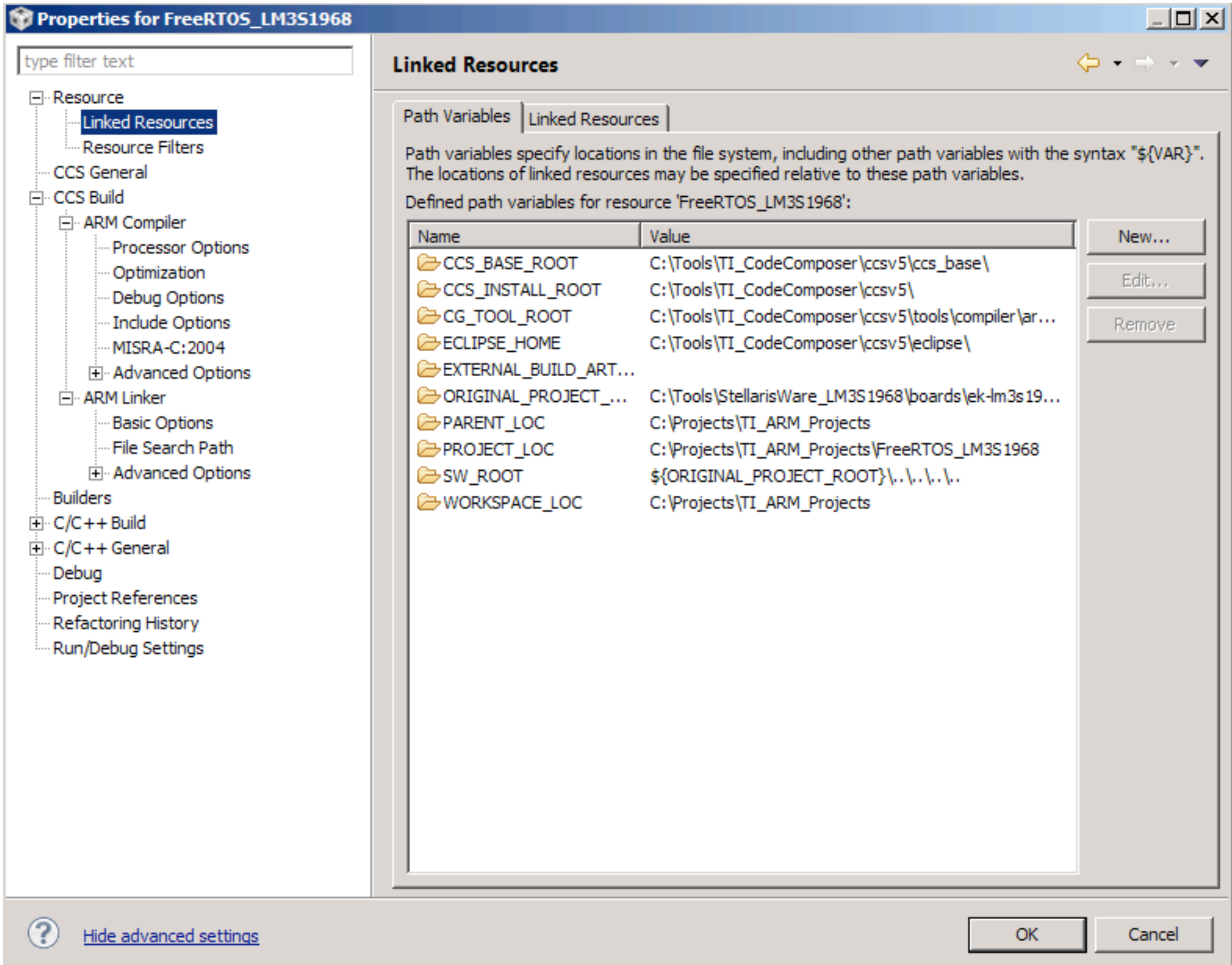
Code Composer -- Library Files



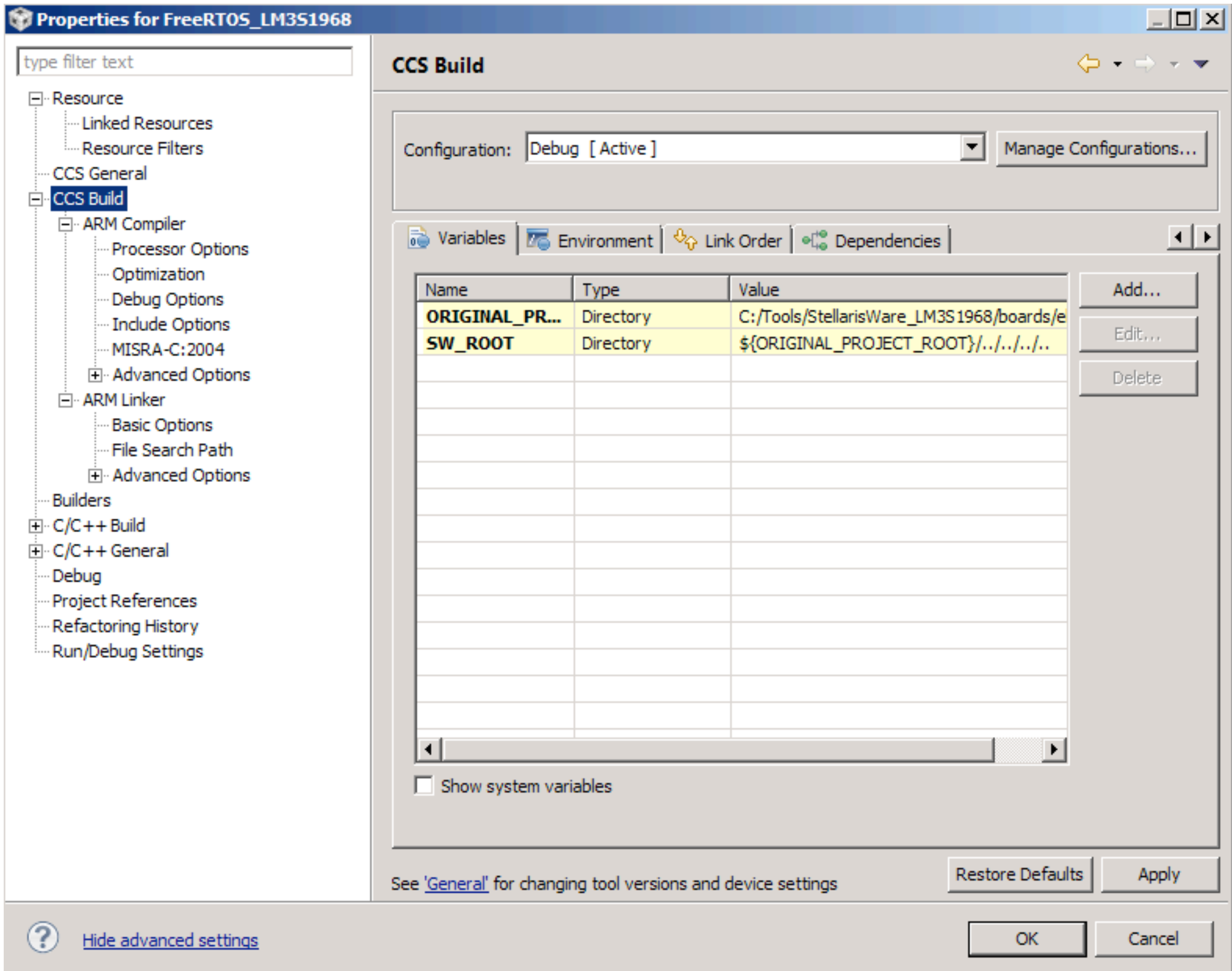
Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

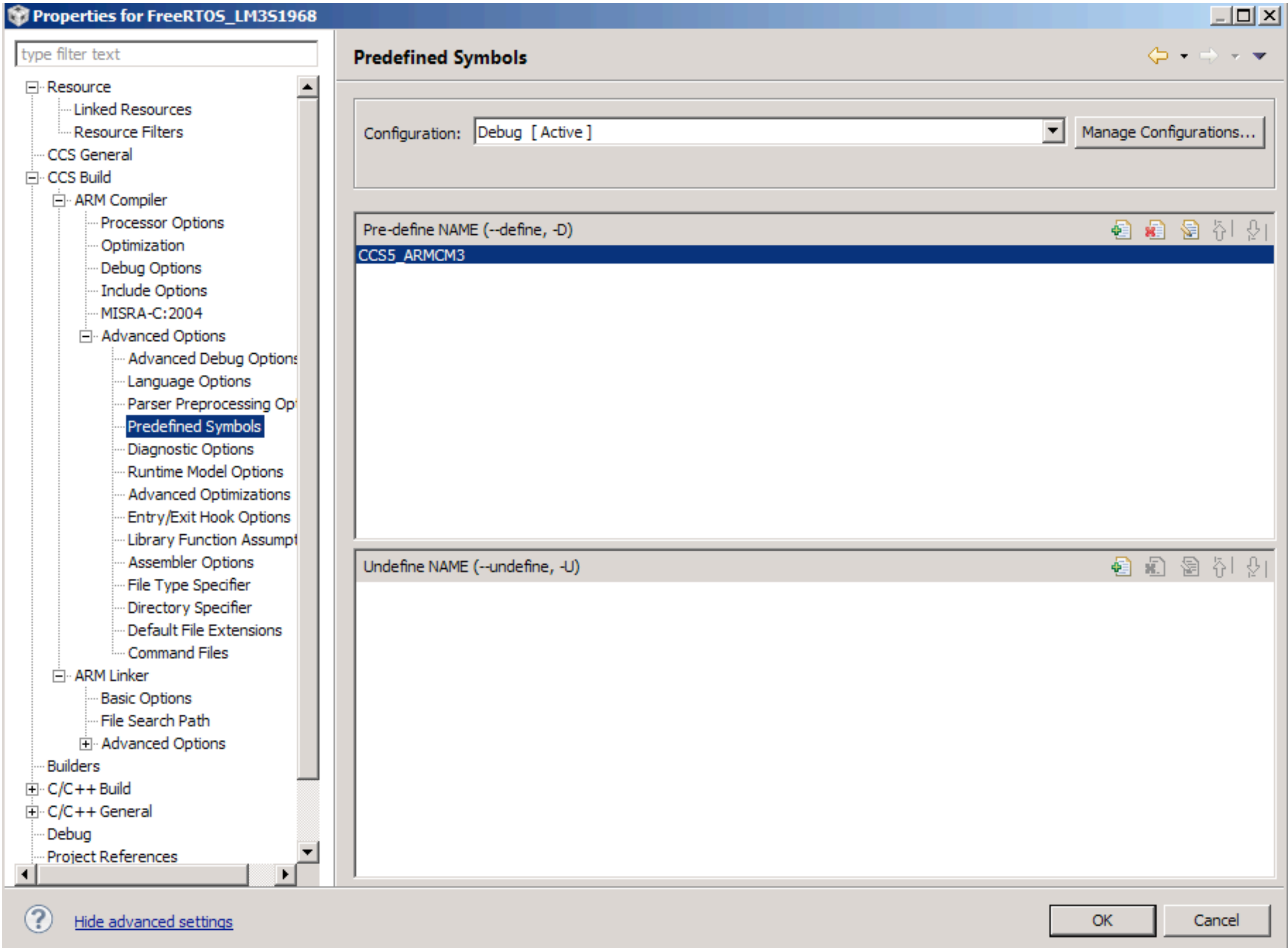
Code Composer -- Linked Resources



Code Composer -- Environment Vars



Code Composer -- Pre-Defined Symbols



FreeRTOSConfig.h

```
/*-----------------------------------------------------------
 * Application specific definitions.
 *
 * These definitions should be adjusted for your particular hardware and
 * application requirements.
 *
 * THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION OF THE
 * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB SITE.
 *
 * See http://www.freertos.org/a00110.html.
 *----------------------------------------------------------*/

#define configUSE_PREEMPTION            0
#define configUSE_IDLE_HOOK         0
#define configUSE_TICK_HOOK         0
#define configCPU_CLOCK_HZ          ( ( unsigned long ) 50000000 )
#define configTICK_RATE_HZ          ( ( portTickType ) 1000 )
#define configMINIMAL_STACK_SIZE        ( ( unsigned short ) 64 )
#define configTOTAL_HEAP_SIZE           ( ( size_t ) ( 8192 ) )
#define configMAX_TASK_NAME_LEN     ( 16 )
#define configUSE_TRACE_FACILITY        0
#define configUSE_16_BIT_TICKS      0
#define configIDLE_SHOULD_YIELD     0
#define configUSE_CO_ROUTINES       0
```

FreeRTOSConfig.h

```
//++GJM
//#define configUSE_TIMERS          1
//++GJM

#define configMAX_PRIORITIES        ( ( unsigned portBASE_TYPE ) 2 )
#define configMAX_CO_ROUTINE_PRIORITIES ( 2 )

#define configKERNEL_INTERRUPT_PRIORITY 255
#define configMAX_SYSCALL_INTERRUPT_PRIORITY 191

/* 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
```

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

Port.c -- Initialize Stack

```
/*-----------------------------------------------------------*/
/*
 * See header file for description.
 */
portSTACK_TYPE *pxPortInitialiseStack( portSTACK_TYPE *pxTopOfStack, pdTASK_CODE pxCode,
                                       void *pvParameters )
{
      /* Simulate the stack frame as it would be created by a context switch interrupt. */
      pxTopOfStack–; /* Offset added to account for the way the MCU uses the stack
                        on entry/exit of interrupts. */
      *pxTopOfStack = portINITIAL_XPSR;    /* xPSR */
      pxTopOfStack–;
      *pxTopOfStack = ( portSTACK_TYPE ) pxCode;    /* PC */
      pxTopOfStack–;
      *pxTopOfStack = 0;  /* LR */
      pxTopOfStack -= 5;  /* R12, R3, R2 and R1. */
      *pxTopOfStack = ( portSTACK_TYPE ) pvParameters;  /* R0 */
      pxTopOfStack -= 8;  /* R11, R10, R9, R8, R7, R6, R5 and R4. */

      return pxTopOfStack;
}
```

Port.c -- xPortSysTickHandler

```
/*-----------------------------------------------------------*/

extern int long xPortSysTickCount = 0;

void xPortSysTickHandler( void ) {
	xPortSysTickCount++;

	/* If using preemption, also force a context switch. */
	#if configUSE_PREEMPTION == 1
	   portNVIC_INT_CTRL_REG = portNVIC_PENDSVSET_BIT;
	#endif

	/* Only reset the systick load register if configUSE_TICKLESS_IDLE is set to
	1.  If it is set to 0 tickless idle is not being used.  If it is set to a
	value other than 0 or 1 then a timer other than the SysTick is being used
	to generate the tick interrupt. */
	#if configUSE_TICKLESS_IDLE == 1
	   portNVIC_SYSTICK_LOAD_REG = ulTimerReloadValueForOneTick;
	#endif

	( void ) portSET_INTERRUPT_MASK_FROM_ISR();
	{
	   vTaskIncrementTick();
	}
	portCLEAR_INTERRUPT_MASK_FROM_ISR( 0 );
}
```

tasks.c -- vTaskIncrementTick

```
void vTaskIncrementTick( void ) {
tskTCB * pxTCB;

      /* Called by the portable layer each time a tick interrupt occurs.
      Increments the tick then checks to see if the new tick value will cause any
      tasks to be unblocked. */
      traceTASK_INCREMENT_TICK( xTickCount );
      if( uxSchedulerSuspended == ( unsigned portBASE_TYPE ) pdFALSE )
      {
         ++xTickCount;
         if( xTickCount == ( portTickType ) 0U )
         {
            xList *pxTemp;

            /* Tick count has overflowed so we need to swap the delay lists.
            If there are any items in pxDelayedTaskList here then there is
            an error! */
            configASSERT( ( listLIST_IS_EMPTY( pxDelayedTaskList ) ) );

            pxTemp = pxDelayedTaskList;
            pxDelayedTaskList = pxOverflowDelayedTaskList;
            pxOverflowDelayedTaskList = pxTemp;
            xNumOfOverflows++;
      .
      .
      .
```

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

FreeRTOS Task States

Task States

- Running -- The task is currently executing
- Ready -- The task can execute and is waiting for the CPU
  - A higher or equal priority task is currently executing
- Blocked -- The task is waiting for an event
  - Temporal Block -- Waiting for an interval or an explicit time
  - Synchronization -- Waiting for an event from another task (e.g. data on a queue) or interrupt
  - Events from Queues, Semaphores, Mutexes, or Interrupts
  - When the “waited for” event occurs, the task moves to the Ready state
- Suspended -- The task is not available for scheduling execution

Task State Transition Diagram

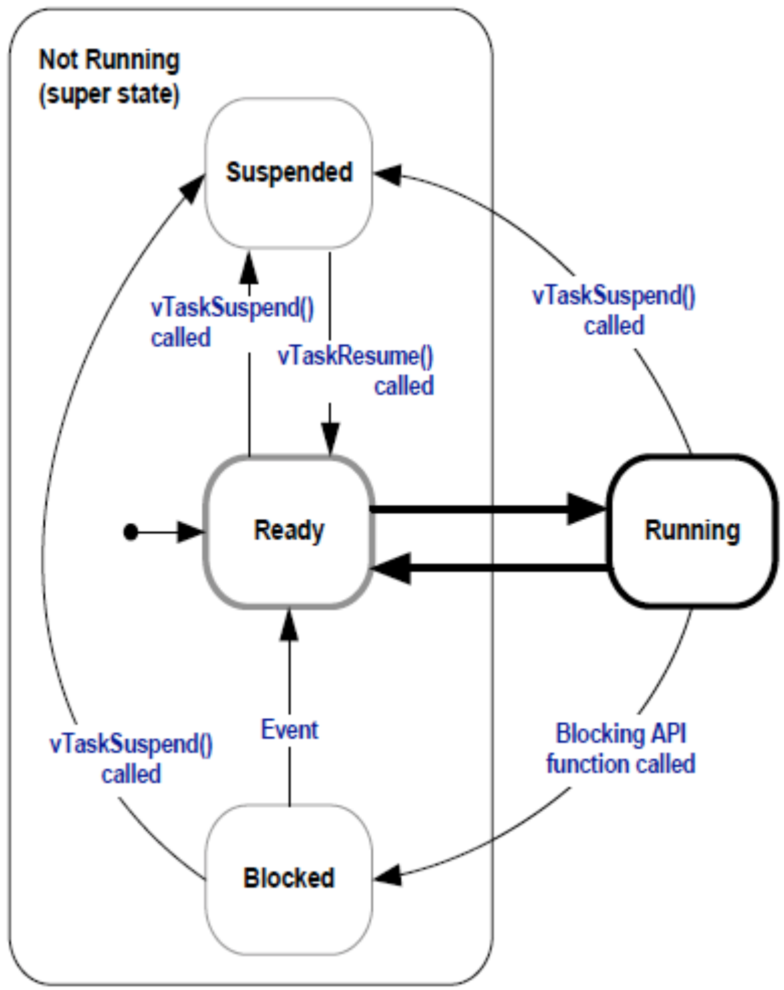


Figure 7. Full task state machine

Example Task -- Print String

```
void vTaskFunction( void *pvParameters )
{
char *pcTaskName;
volatile unsigned long ul;

    /* The string to print out is passed in via the parameter.  Cast this to a
    character pointer. */
    pcTaskName = ( char * ) pvParameters;

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );

        /* Delay for a period. */
        for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++ )
        {
            /* This loop is just a very crude delay implementation.  There is
            nothing to do in here.  Later exercises will replace this crude
            loop with a proper delay/sleep function. */
        }
    }
}
```

Listing 8. The single task function used to create two tasks in Example 2

Example Task -- Set-up Tasks

```
/* Define the strings that will be passed in as the task parameters.  These are
defined const and not on the stack to ensure they remain valid when the tasks are
executing. */
static const char *pcTextForTask1 = "Task 1 is running\n";
static const char *pcTextForTask2 = "Task 2 is running\n";

int main( void )
{
    /* Create the first task at priority 1.  The priority is the second to last
    parameter. */
    xTaskCreate( vTaskFunction, "Task 1", 240, (void*)pcTextForTask1, 1, NULL );

    /* Create the second task at priority 2. */
    xTaskCreate( vTaskFunction, "Task 2", 240, (void*)pcTextForTask2, 2, NULL );

    /* Start the scheduler so the tasks start executing. */
    vTaskStartScheduler();

    /* If all is well we will never reach here as the scheduler will now be
    running.  If we do reach here then it is likely that there was insufficient
    heap available for the idle task to be created. */
    for( ;; );
}
```

Listing 10. Creating two tasks at different priorities

Example Task -- Execution

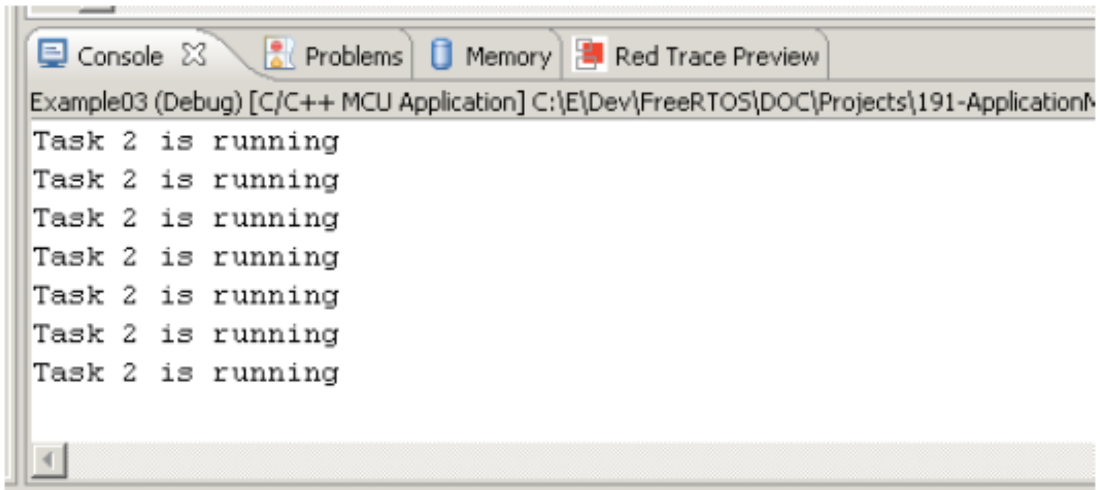


Figure 5. Running both test tasks at different priorities

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

Example Task -- Print Task w/ Delay


```
void vTaskFunction( void *pvParameters )
{
char *pcTaskName;

    /* The string to print out is passed in via the parameter.  Cast this to a
    character pointer. */
    pcTaskName = ( char * ) pvParameters;

    /* As per most tasks, this task is implemented in an infinite loop. */
    for( ;; )
    {
        /* Print out the name of this task. */
        vPrintString( pcTaskName );

        /* Delay for a period.  This time a call to vTaskDelay() is used which
        places the task into the Blocked state until the delay period has expired.
        The delay period is specified in 'ticks', but the constant
        portTICK_RATE_MS can be used to convert this to a more user friendly value
        in milliseconds.  In this case a period of 250 milliseconds is being
        specified. */
        vTaskDelay( 250 / portTICK_RATE_MS );
    }
}
```

Listing 12. The source code for the example task after the null loop delay has been replaced by a call to vTaskDelay()

Example Task -- Execution w/ Delay

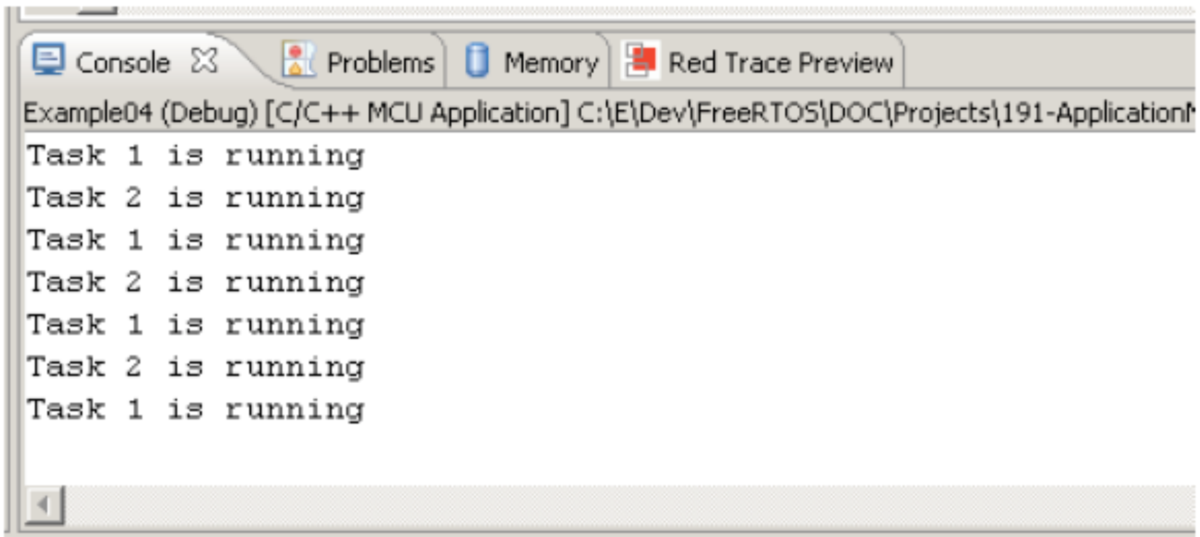


Figure 8. The output produced when Example 4 is executed

Information and Telecommunication Technology Center

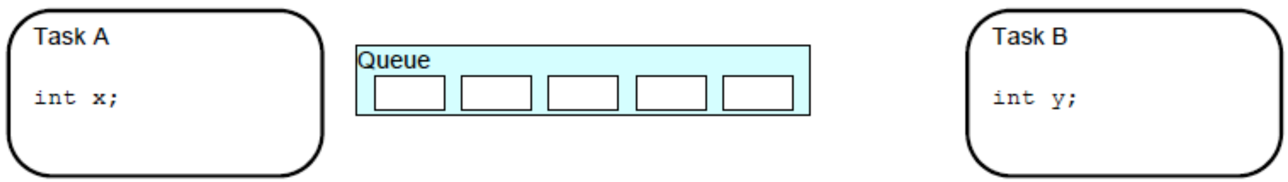
THE UNIVERSITY OF KANSAS

FreeRTOS Queues

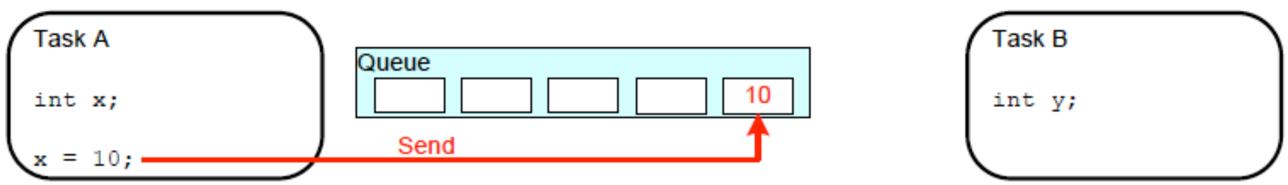
Queues

- Queues hold a finite number of items
  - Number of items and item size determined at queue create time
- Queues implement a First-In/First-Out (FIFO) protocol
- Sending/Receiving items are by copy not reference
- Queues are system-wide resources
- Queue Functions
  - Create Queues
  - Send/Receive Data to/from Queues
  - Queue Management/Number of Items in Queue
  - Blocking on a Queue/Effect of Priority

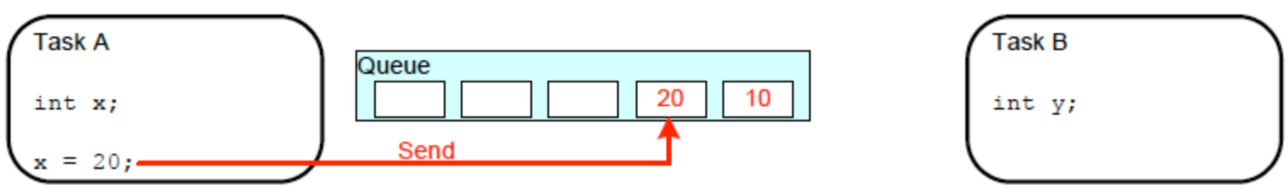
Queue Example



A queue is created to allow Task A and Task B to communicate. The queue can hold a maximum of 5 integers. When the queue is created it does not contain any values so is empty.



Task A writes (sends) the value of a local variable to the back of the queue. As the queue was previously empty the value written is now the only item in the queue, and is therefore both the value at the back of the queue and the value at the front of the queue.



Task A changes the value of its local variable before writing it to the queue again. The queue now contains copies of both values written to the queue. The first value written remains at the front of the queue, the new value is inserted at the end of the queue. The queue has three empty spaces remaining.

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

Queue -- Send Data

```
portBASE_TYPE xQueueSendToFront(   xQueueHandle xQueue,
                                   const void * pvItemToQueue,
                                   portTickType xTicksToWait
                                 );
```

Listing 30. The xQueueSendToFront() API function prototype

```
portBASE_TYPE xQueueSendToBack(    xQueueHandle xQueue,
                                   const void * pvItemToQueue,
                                   portTickType xTicksToWait
                                 );
```

Listing 31. The xQueueSendToBack() API function prototype

xTicksToWait -- number of ticks to wait if queue is full
pdPASS -- return value if en-queued
errQUEUE_FULL -- return value if queue is full

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

Queue -- Receive Data

```
portBASE_TYPE xQueueReceive(
                        xQueueHandle xQueue,
                        const void * pvBuffer,
                        portTickType xTicksToWait
                    );
```

Listing 32. The xQueueReceive() API function prototype

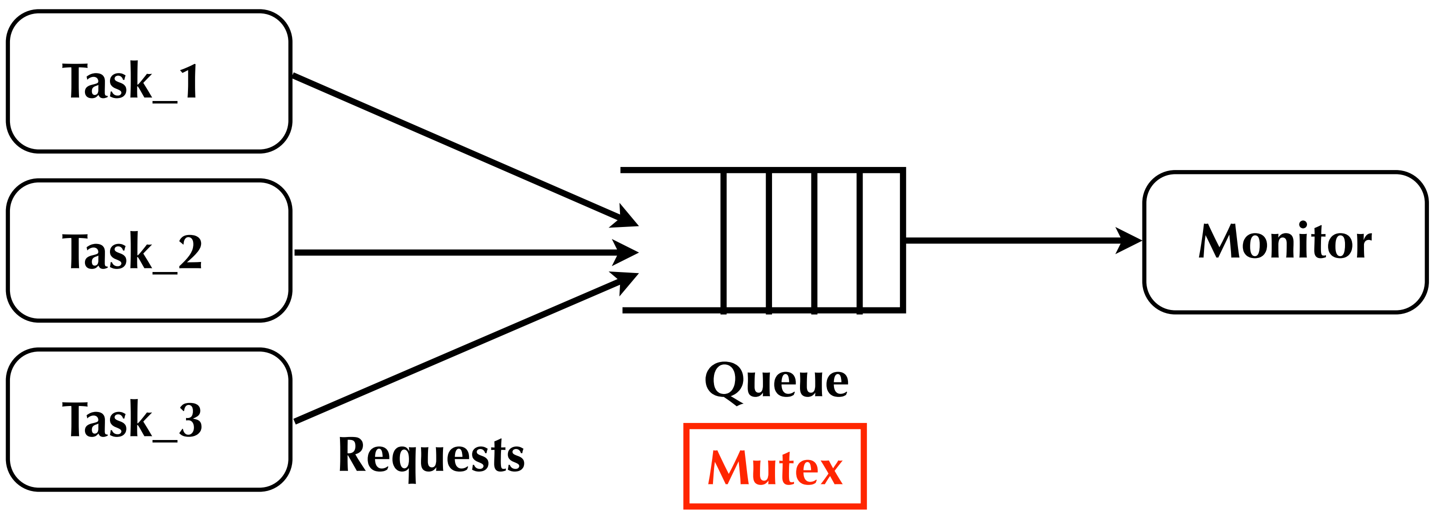
```
portBASE_TYPE xQueuePeek(
                        xQueueHandle xQueue,
                        const void * pvBuffer,
                        portTickType xTicksToWait
                    );
```

Listing 33. The xQueuePeek() API function prototype

xQueuePeek -- does not remove item from queue
xTicksToWait -- number of ticks to wait if queue is empty
pdPASS -- return value if en-queued
errQUEUE_EMPTY -- return value if queue is empty

Using Queues

- A single task, the Monitor, manages the shared resource
- Requests to read or write data are queued for the Monitor
    - Requests can be prioritized
- Monitor processes one request at a time
- Access to queue managed by mutex



FreeRTOS Mutual Exclusion

Mutual Exclusion

- Problem Description
- FreeRTOS Support

Multiple Tasks Access Common Data

```
extern volatile long int SysTickCount_Low;        // Low 32-bits of SysTickCount
extern volatile long int SysTickCount_High;       // High 32-bits of SysTickCount

=========================================================================================

void SysTickISR() {
      SysTickCount_Low++;                          // Increment low 32-bits
      if (SysTickCount_Low == 0 ) {                // Check for wrap-around
          SysTickCount_High++;                     // Increment high 32-bits
      }
}

=========================================================================================

         .
         .
long int Time_Low, Time_High;                      // Local time value
         .
         .
         .
      Time_Low = SysTickCount_Low;                 // Copy SysTickCount_Low
      Time_High = SysTickCount_High;               // Copy SysTickCount_High
         .
         .
         .
```

Interrupt!

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

Task Execution Sequence

Time_High	Time_Low	SysTickCount_High	SysTickCount_Low	
		0x0000 0000	0xFFFF FFFF	
	0xFFFF FFFF			Copy Low
	0xFFFF FFFF			Interrupt!
	0xFFFF FFFF	0x0000 0000	0x0000 0000	Incr./Wrap Low
	0xFFFF FFFF	0x0000 0001	0x0000 0000	Incr. High
0x0000 0001	0xFFFF FFFF			Copy High

Multiple Tasks Access Common Data

```
extern volatile long int SysTickCount_Low;          // Low 32-bits of SysTickCount
extern volatile long int SysTickCount_High;         // High 32-bits of SysTickCount

=========================================================================================

void SysTickISR() {
      SysTickCount_Low++;                           // Increment low 32-bits
}

=========================================================================================

         .
         .
long int Time_Low, Time_High;                       // Local time value
         .
         .
         .
      Time_Low = SysTickCount_Low;                  // Copy SysTickCount_Low

         .
         .
         .
```

Early Interrupt!

Late Interrupt!

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

Task Execution Sequence -- Early Inter.

Time_Low	SysTickCount_Low	
	0xFFFF FFFF	
		Early Interrupt!
	0x0000 0000	Incr./Wrap Low
0x0000 0000		Copy Low

Task Execution Sequence

Time_Low	SysTickCount_Low	
	0xFFFF FFFF	
0xFFFF FFFF		Copy Low
0xFFFF FFFF		Interrupt!
0xFFFF FFFF	0x0000 0000	Incr./Wrap Low

Information and Telecommunication Technology Center

THE UNIVERSITY OF KANSAS

Mutual Exclusion -- Conditions

- Shared Data or Hardware Resource among multiple Tasks
  - E.g. combined **`SysTickCount_Low`** and **`SysTickCount_High`**
  - E.g. UART, ADC, ...
  - Exclusive resource access does not cause problems
- Non-atomic access

```
Time_Low = SysTickCount_Low;              // Copy SysTickCount_Low
Time_High = SysTickCount_High;            // Copy SysTickCount_High
```

- Multiple Processors

Task_1	Task_2
`myNbrAvailWidgets = NbrAvailWidgets;`	
	`myNbrAvailWidgets = NbrAvailWidgets;`
	`NbrAvailWidgets = myNbrAvailWidgets – 100;`
`NbrAvailWidgets = myNbrAvailWidgets – 100;`	

Mutual Exclusion -- Approaches

- Identify Critical Sections -- Multiple tasks access a (non-atomic) shared resource, e.g. extended SysTickCount
- Protect critical sections with a lock/un-lock shared resource actions

Mutual Exclusion -- Inhibit Interrupts

- Single Processor -- Inhibit Interrupts
  - Prevents scheduling another task or interrupt during critical section

```
taskENTER_CRITICAL();                            // Enter critical section

Time_Low = SysTickCount_Low;                     // Copy SysTickCount_Low
Time_High = SysTickCount_High;                   // Copy SysTickCount_High

taskEXIT_CRITICAL();                             // Exit critical section
```

- Can make the system unresponsive
  - Protects the entire system... not just the shared resource
  - Interrupts not handled in a timely manner
  - High priority tasks not scheduled and executed
- Does not work in multi-processor/multi-core when any processor can handle interrupts

Mutual Exclusion -- Mutex

- Mutual Exclusion Control Variable
  - Mutex associated with shared resource
  - Task must "own" Mutex to use shared resource
  - Task must "release Mutex when finished with shared resource
  - Task may "queue" waiting for Mutex
  - Requires hardware support to "test-and-set" Mutex Variable with an atomic instruction

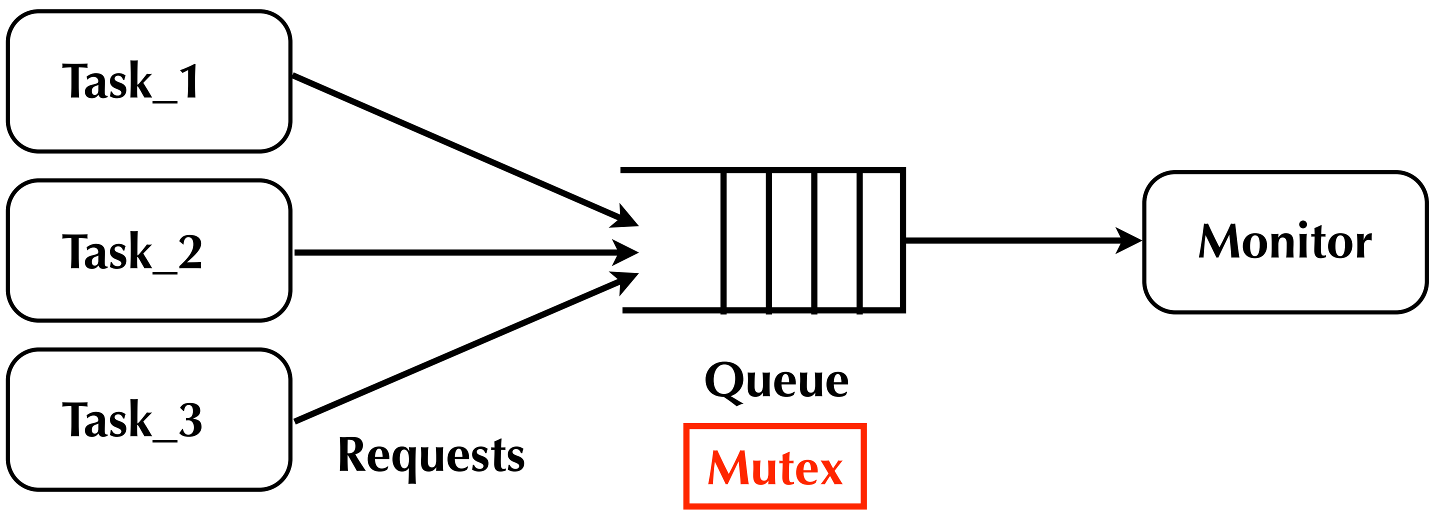
```
xMyMutexHandle = xSemaphoreCreateMutex();     // Create in main.c and make global

xSemaphoreTake( xMyMutexHandle, portMaxDelay );     // Acquire Mutex;
                                                    // Wait if not available

Time_Low = SysTickCount_Low;                  // Copy SysTickCount_Low
Time_High = SysTickCount_High;                // Copy SysTickCount_High

xSemaphoreGive( xMyMutexHandle);              // Exit critical section
```

Mutual Exclusion -- Compare-and-Set

- As an atomic operation
  - Read current mutex variable
  - Compare to 0
  - If 0, set to one; exit
  - Loop
- Spin-Loop
- Uses CPU resources
- Requires hardware, CPU, cache, and memory system, support
- No queuing of requests

Mutual Exclusion -- Queue and Monitor

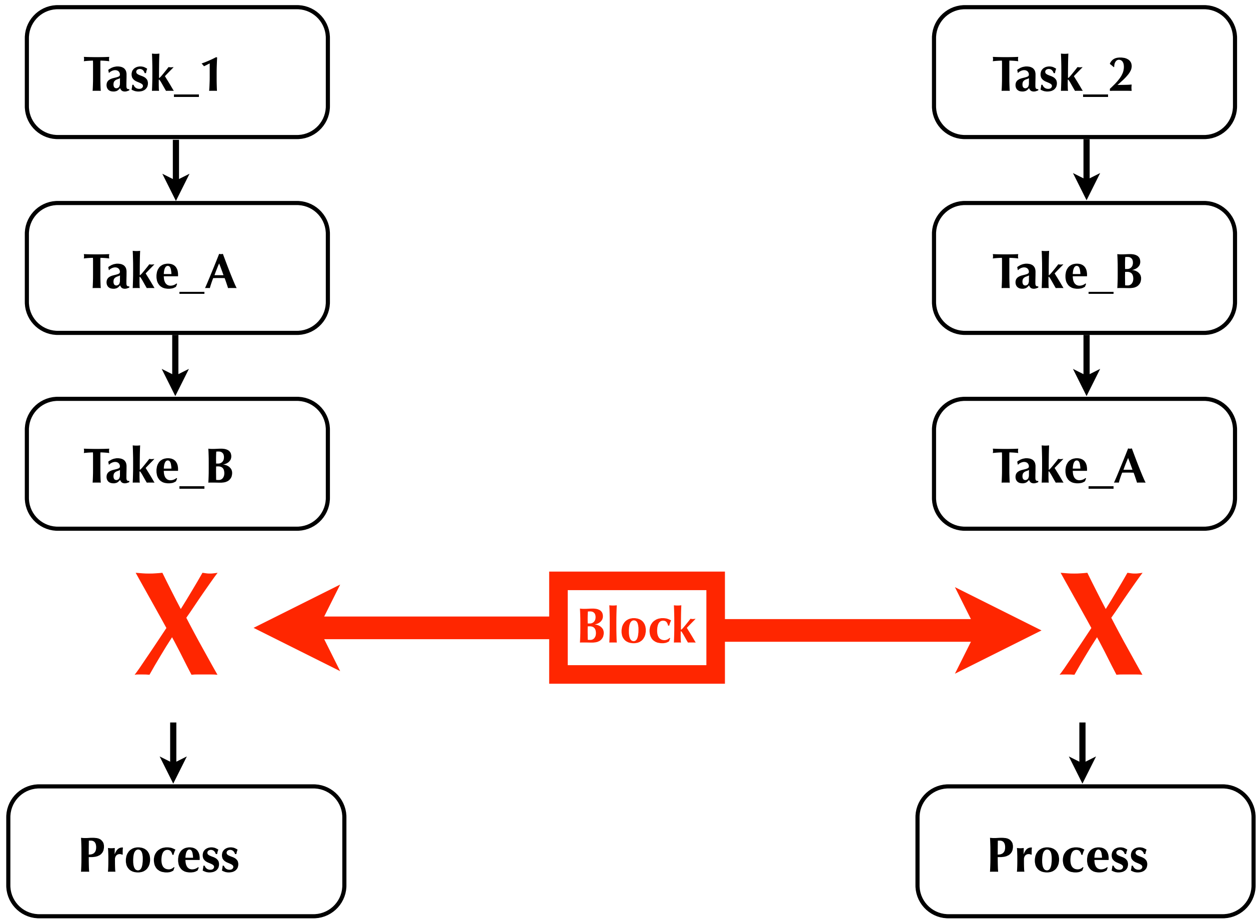
- A single task, the Monitor, manages the shared resource
- Requests to read or write data are queued for the Monitor
  - Requests can be prioritized
- Monitor processes one request at a time
- Access to queue managed by mutex



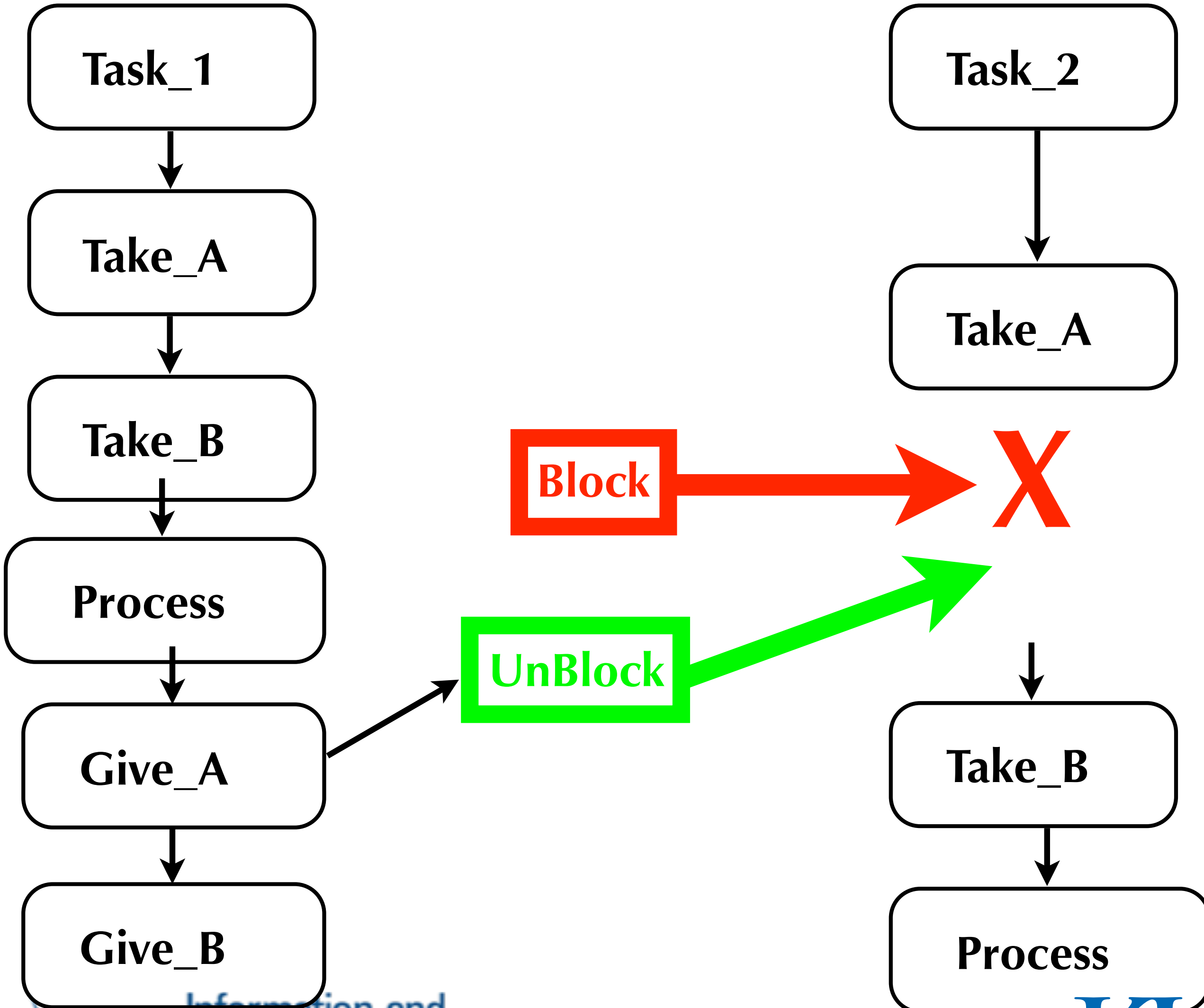
Mutual Exclusion -- Problems

- Deadlock
- Priority Inversion

Mutual Exclusion -- Deadlock



Mutual Exclusion -- Deadlock Avoidance



Mutual Exclusion -- Priority Inversion