# COE718: Embedded Systems Design

## Lecture 5:
## Scheduling with RTX/CMSIS

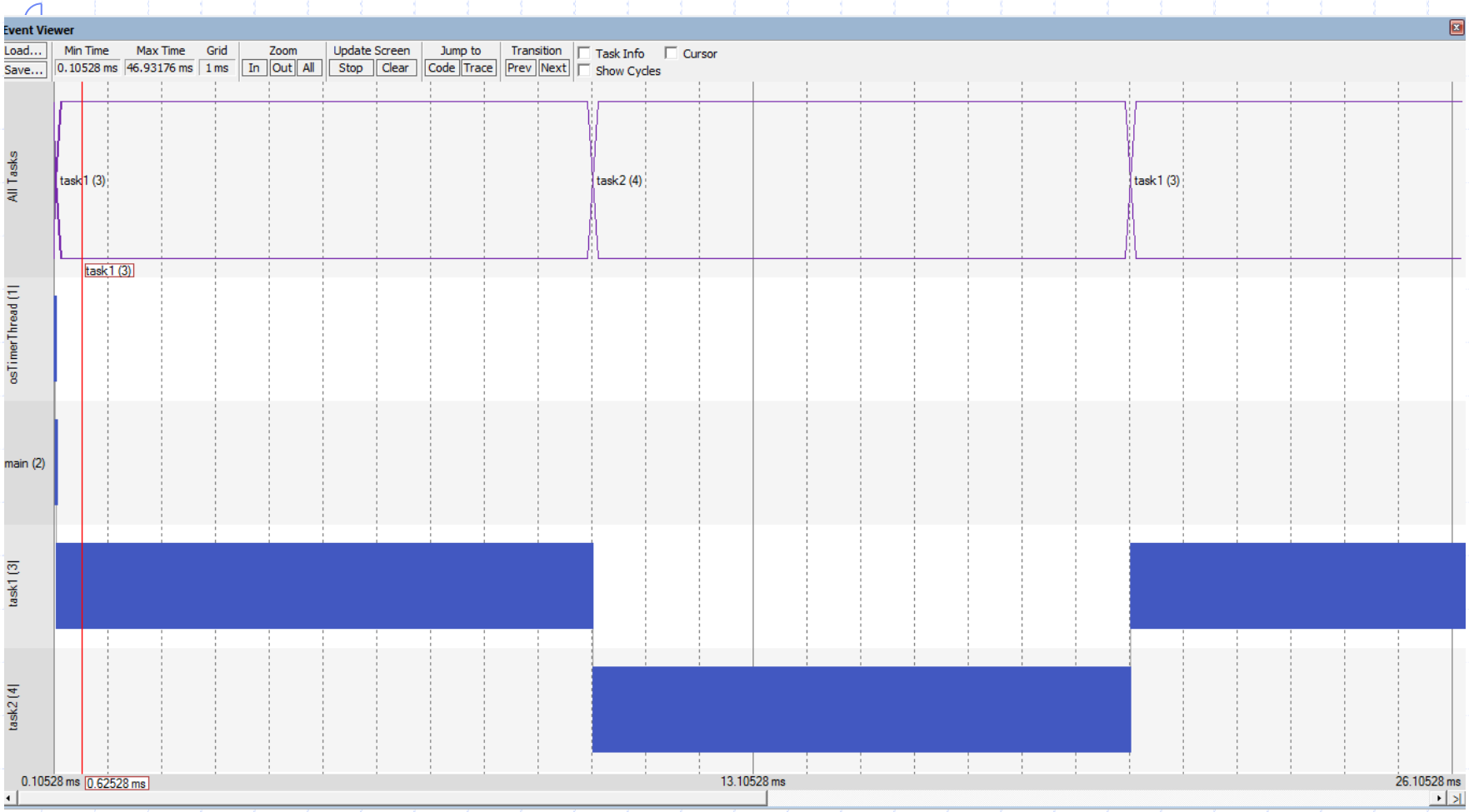
Anita Tino

# Round Robin Scheduling (contd)

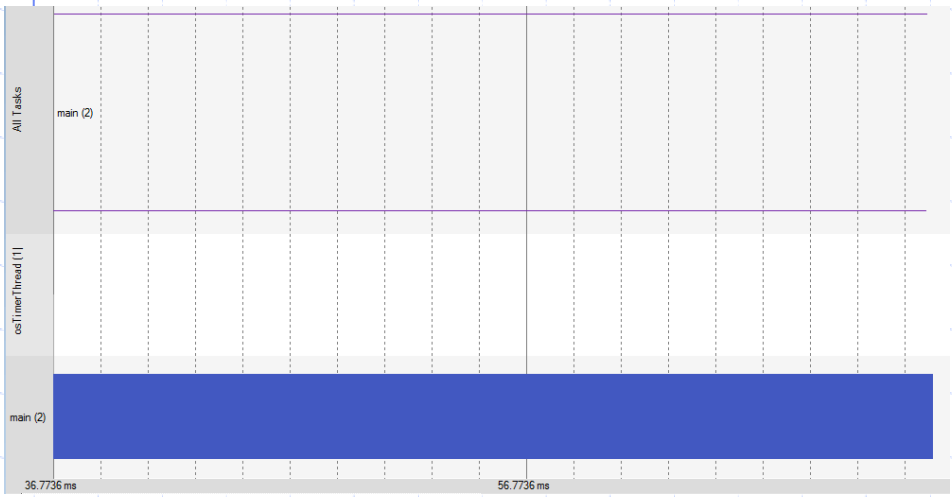
```
#include <stdio.h>
#include "LPC17xx.h"
#include <RTL.h>

long global_c1 = 0, global_c2 = 0;

__task void task1(void){
    for(;;){
        global_c1 += 3;
    }
}
__task void task2(void){
    for(;;){
        global_c2 += 2;
    }
}
```

```
int main(void){
    SystemInit();
    os_tsk_create(task1, 1);
    os_tsk_create(task2, 1);
    os_tsk_delete_self();

    os_sys_init(task1);
}
```

# Round Robin Scheduling (contd)



# Round Robin Scheduling (contd)




```
int main(void){
    SystemInit();
    os_tsk_create(task1, 1);
    os_tsk_create(task2, 1);
    os_sys_init(task1);

    os_tsk_delete_self();
}
```

# Cooperative Multitasking

```
#include <stdio.h>
#include "LPC17xx.h"
#include <RTL.h>

int counter1, counter2;
__task void task1(void);
__task void task2(void);

__task void task1(void){
    for(;;){
        counter1++;
        os_tsk_pass();
    }
}
```

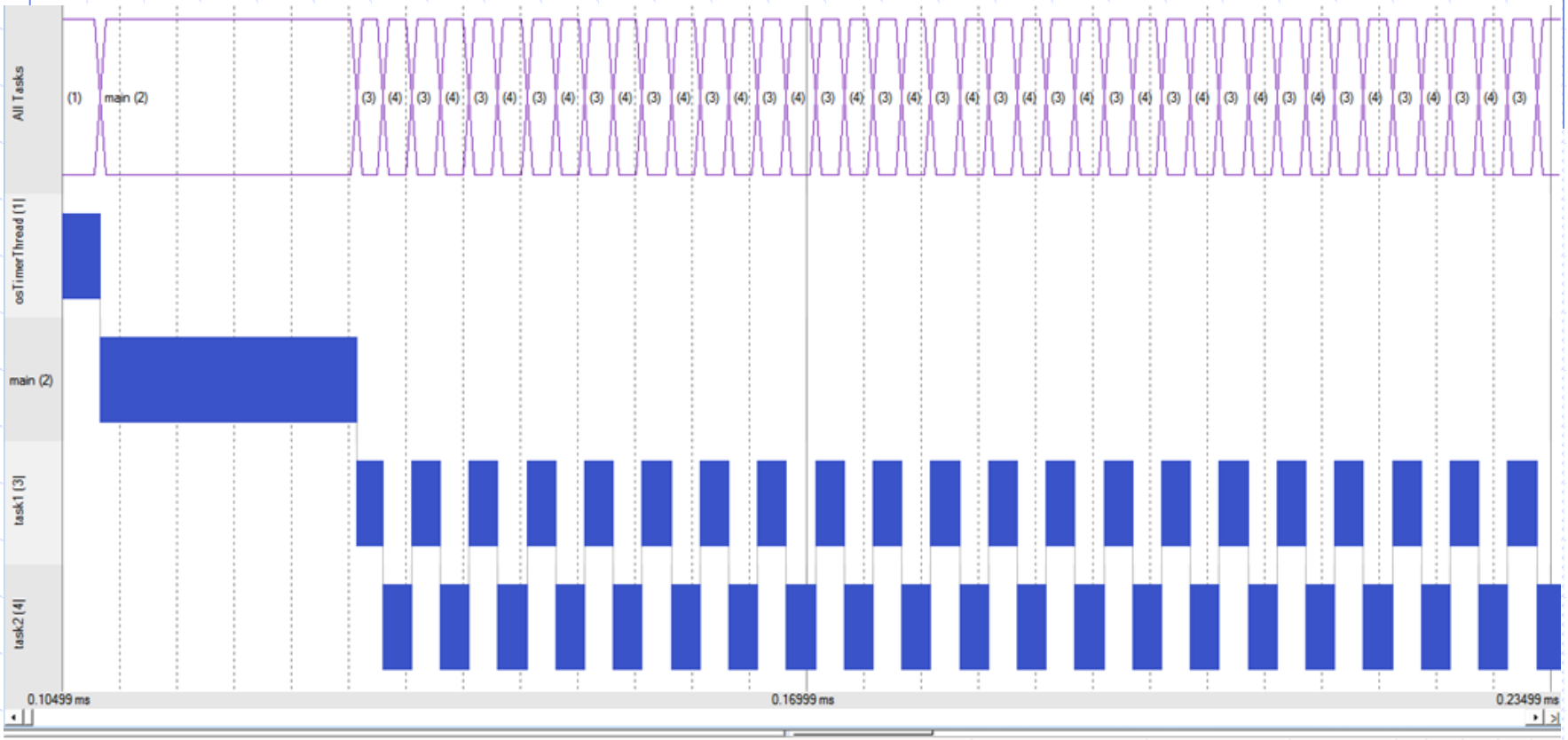
```
__task void task2(void){
    for(;;){
        counter2++;
        os_tsk_pass();
    }
}

int main(void){
    os_sys_init(task1);
    for(;;);
}
```

# Cooperative Multitasking

```
#include <stdio.h>
#include "LPC17xx.h"
#include <RTL.h>

int counter1, counter2;
__task void task1(void);
__task void task2(void);

__task void task1(void){
    for(;;){
        counter1++;
        os_tsk_pass();
    }
}
```

```
__task void task2(void){
    for(;;){
        counter2++;
        os_tsk_pass();
    }
}

int main(void){
    SystemInit();
    os_tsk_create(task1, 1);
    os_tsk_create(task2, 1);
    os_tsk_delete_self();

    os_sys_init(task1);
}
```

# Cooperative Multitasking



# RTX Functions/Definitions

- os_tsk_pass() -Pass control
- os_tsk_delete_self() -Delete self
- os_mut_init() -Initialize mutex
- os_mut_wait() -Wait for mutex availability
- os_mut_release() -Release the mutex
- os_evt_set(flag, tid) -Set a flag (synchronize)
- os_evt_wait_and(..) -Wait for all flags
- os_dly_wait(ticks) -puts task in wait state

# RTX Functions/Definitions

- os_itv_wait()
- os_sem_init()
- os_mbx_wait()
- os_mbx_send()
- os_tsk_prio_self()
- os_tsk_self()

-Wait for certain time interval

-Initialize semaphore

-Wait for mailbox post

-Post a message to mailbox

-Change task's priority level

- Get task's self priority level

# Pre-emptive Scheduling

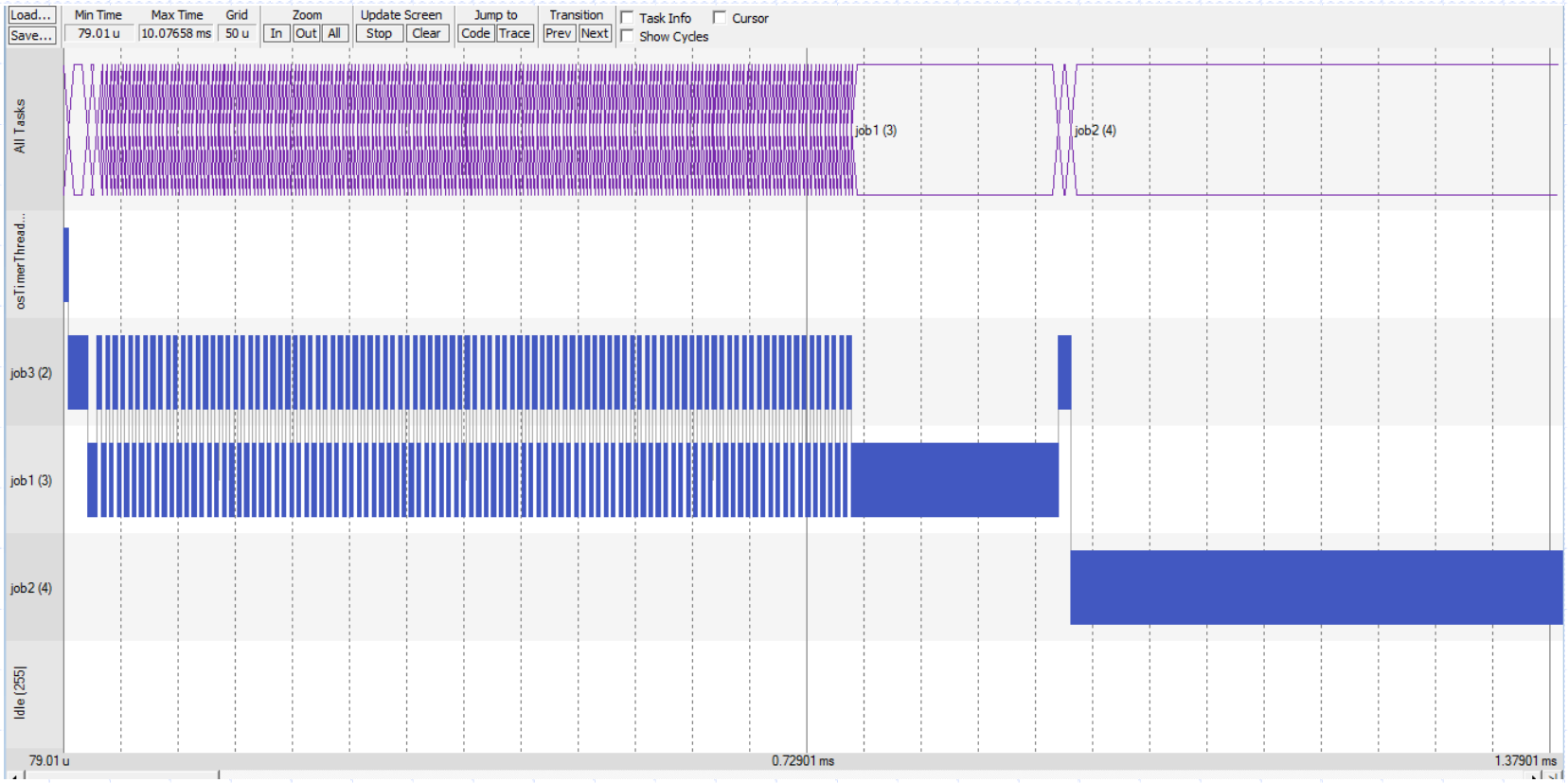
```
#include <stdio.h>
#include "LPC17xx.h"
#include <RTL.h>

OS_TID id1, id2;
double  cnt1,cnt2, cnt3;

__task void job1 (void);
__task void job2 (void);
__task void job3 (void);

__task void job1 (void) {
     id1 = os_tsk_self(); //identify myself and create job 3
     id2 = os_tsk_create(job3, 0x0); //create and return

     for(;;){
       if(cnt1 < 200){
            cnt1++;
            os_tsk_pass(); //increment alongside job 3 by passing token (same priority)
        }else{
            os_evt_set(0x0004, id2); //finished counting. Signal to job3
       }
     }
}
```

# Pre-emptive Scheduling

```
__task void job2 (void) {
    for(;;){
            if(cnt2 < 700){ //increment to 700 and delete
                        cnt2++;
                        os_tsk_pass();
            }else{
                        os_tsk_delete_self(); //should go to idle_deamon after this
            }
    }

}
__task void job3 (void) {
    id2 = os_tsk_self(); //obtain my identity
    for(;;){
       if(cnt3 < 100){
         cnt3++; //keep incrementing and passing token to job1
          os_tsk_pass();
        }else if(cnt3 >= 100){
          os_tsk_prio_self(0x05);
           //once finished, increase my priority so that I may get job1's signal
          if(os_evt_wait_and(0x0004, 0xFFFF)){
              os_tsk_create(job2, 0x0);
              //and create job 2 while deleting job 1 and myself
              os_tsk_delete(id1);
              os_tsk_delete_self();
            }
         }
    }
}
```

```
int main (void) {
     os_tsk_create(job1, 0x0);
    //create job1 and initialize
     SystemInit();
     os_tsk_delete_self();
}
```

# Pre-emptive Scheduling



# Timing Specifications

| Function | ARM7™/ARM9™ (cycles) | Cortex™-M (cycles) |
|---|---|---|
| Initialize system, (*os_sys_init*), start task | 1721 | 1147 |
| Create task (no task switch) | 679 | 403 |
| Create task (switch task) | 787 | 461 |
| Delete task (*os_tsk_delete*) | 402 | 218 |
| Task switch (by *os_tsk_delete_self*) | 458 | 230 |
| Task switch (by *os_tsk_pass*) | 321 | 192 |
| Set event (no task switch) | 128 | 89 |
| Set event (switch task) | 363 | 215 |
| Send semaphore (no task switch) | 106 | 72 |
| Send semaphore (switch task) | 364 | 217 |
| Send message (no task switch) | 218 | 117 |
| Send message (switch task) | 404 | 241 |
| Get own task identifier (*os_tsk_self*) | 23 | 65 |
| Interrupt lockout | <160 | 0 |

# A bit on Mutexes

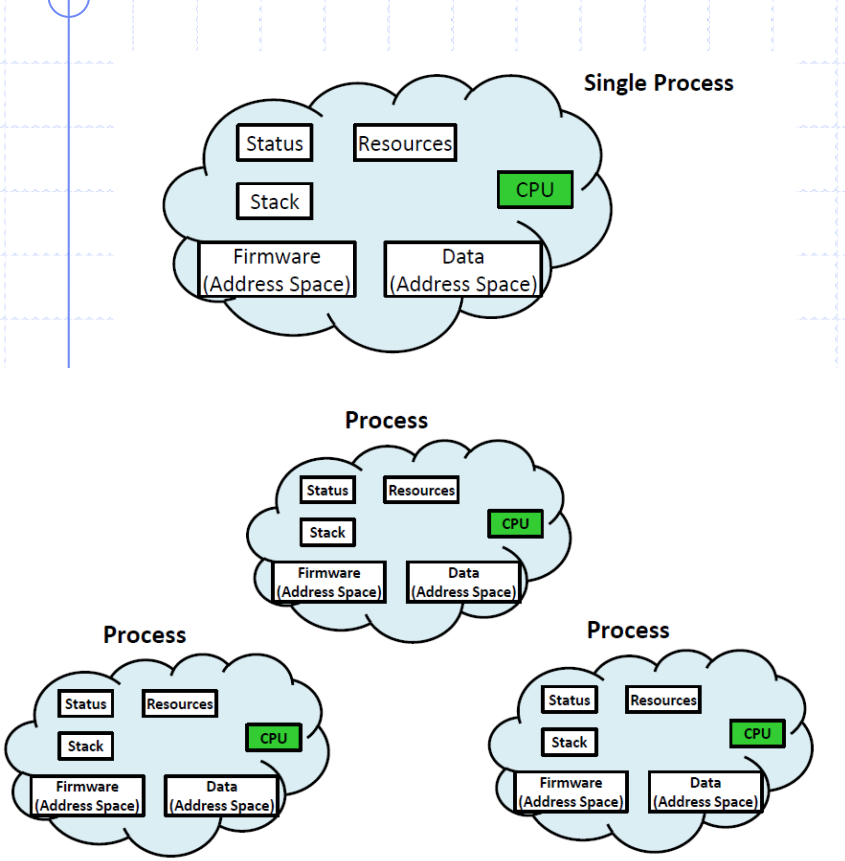
```
#include <stdio.h>
#include "LPC17xx.h"
#include <RTL.h>

__task void mem(void);
__task void appl(void);

OS_MUT mut1;
char logger[100]; //atomic var

__task void appl(void){
    ...
    //need to access critical
   section
   os_mut_wait(&mut1, 0xFFFF);
   logger[..] = ....
   os_mut_release(&mut1);
}
```

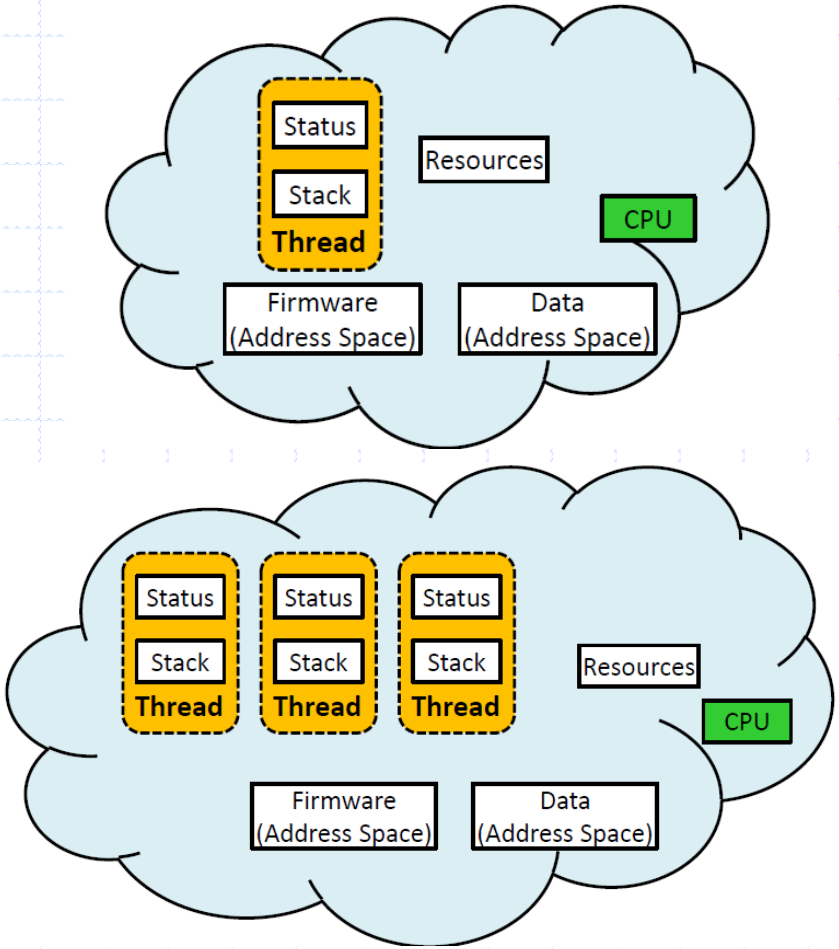
```
__task void mem(void){
    ...
    //need to access critical
   section
   os_mut_wait(&mut1, 0xFFFF);
   logger[..] = ....
   os_mut_release(&mut1);
}

int main(void){
   ....
   os_mut_init(&mut1);
   ....
}
```

# Threads Executing Tasks

- A process/task is a collection of resources (copy of regs, PC, stack) that are utilized to execute a program
- Thread - The smallest subset of these resources necessary to exe the program
  - Unit of computation with code and context, but no private data
  - Can only be in 1 process

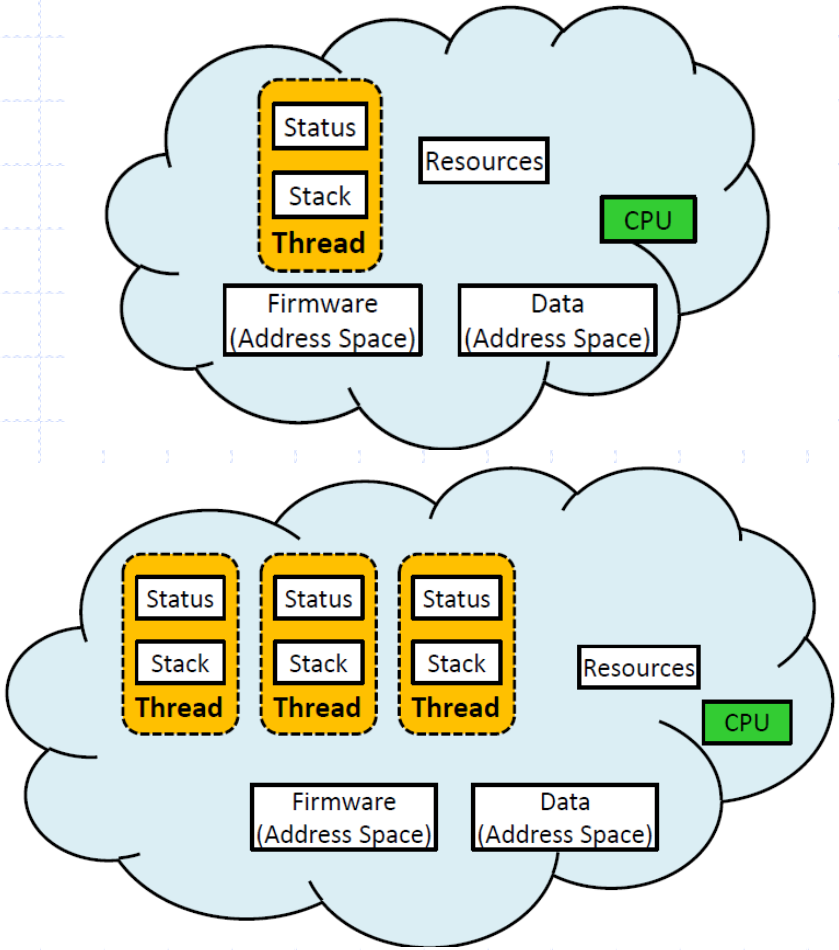
# Processes versus Threads



# Processes versus Threads

- Process divided into subjobs (i.e. threads of execution)
- Threads not independent of each other
  - Run in shared memory space
  - Don't need explicit inter-process communication
  - Lightweight process

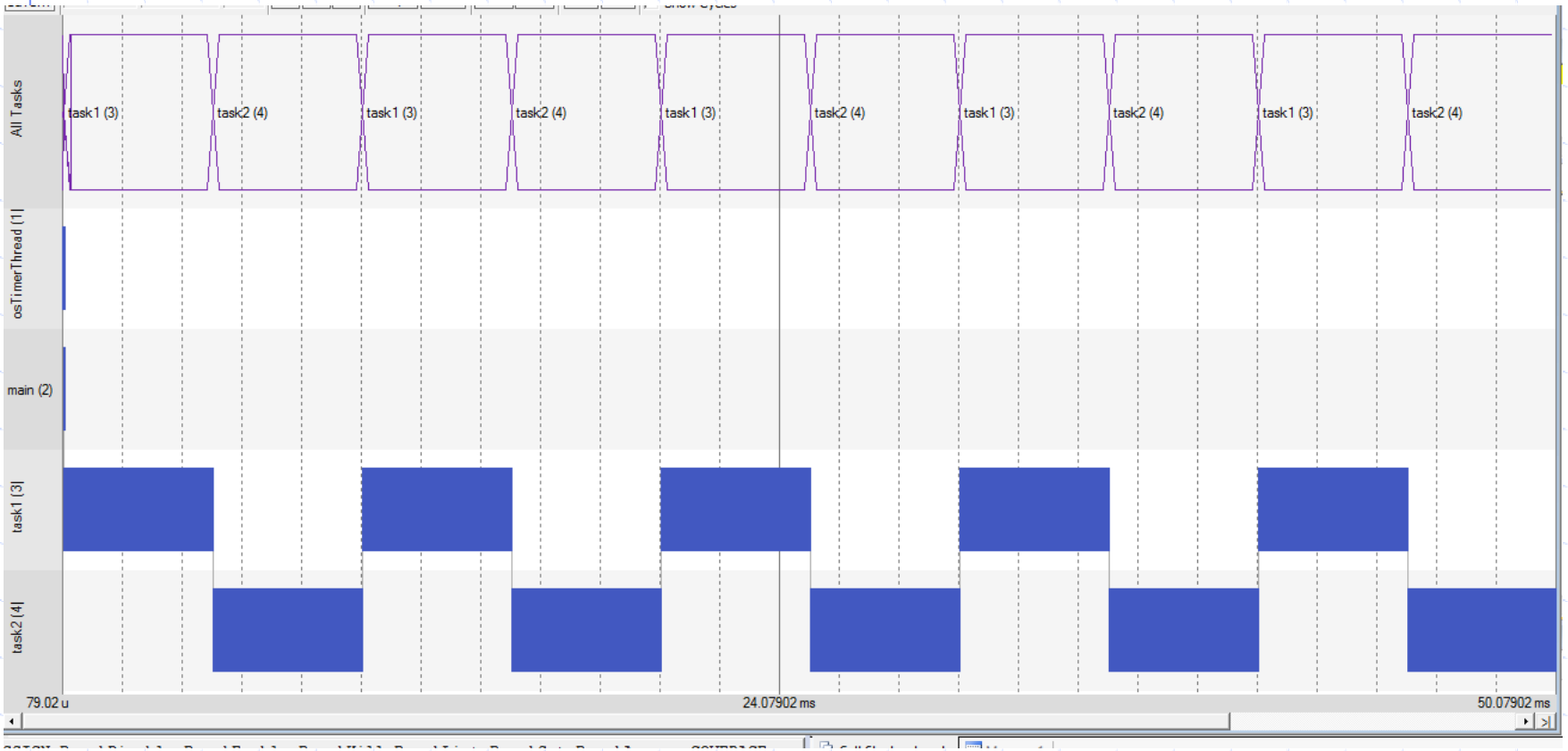


# Multithreading Example

```
#include "cmsis_os.h"
#include "RTL.H"
#include "LPC17xx.H"

//global task counters
unsigned int counta = 0;
unsigned int countb = 0;

__task void task1 (void const *arg) {
    for (;;)  // Infinite loop - runs
    while task1 runs.
          counta++;
}
__task void task2 (void const *arg) {
    for (;;)
          countb++;
}
```

```
//create a thread for the function task1 and
    task2 with normal priority
osThreadDef (task1, osPriorityNormal, 1, 0);
osThreadDef (task2, osPriorityNormal, 1, 0);
//osThreadDef(name, priority, num of instances, stack
    size);

int main (void) {
    SystemInit();
    osKernelInitialize ();
    // setup kernel to create os* objects
    osThreadCreate (osThread(task1), NULL);
    // create threads
    osThreadCreate (osThread(task2), NULL);
    osKernelStart ();
    // start kernel

    osDelay(osWaitForever);
}
```

# Multithreading Example



# CMSIS Thread Priority Levels

osPriorityIdle
osPriorityLow
osPriorityBelowNormal
osPriorityNormal
osPriorityAboveNormal
osPriorityHigh
osPriorityRealTime
osPriorityError

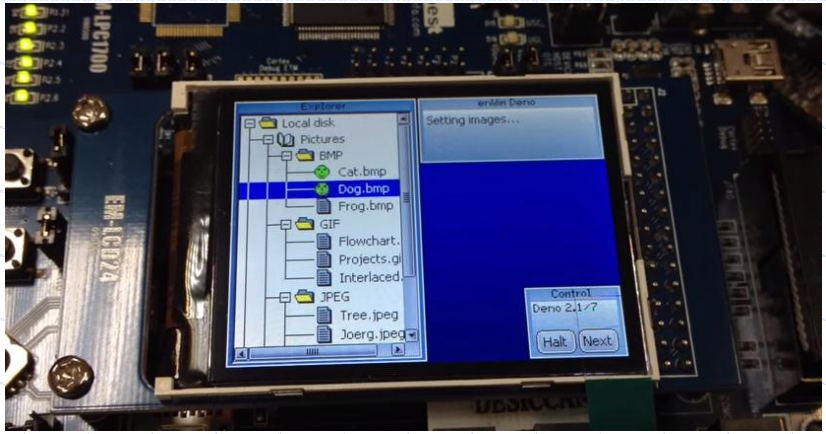
```
osThreadDef (task1, osPriorityNormal, 1, 0);
```

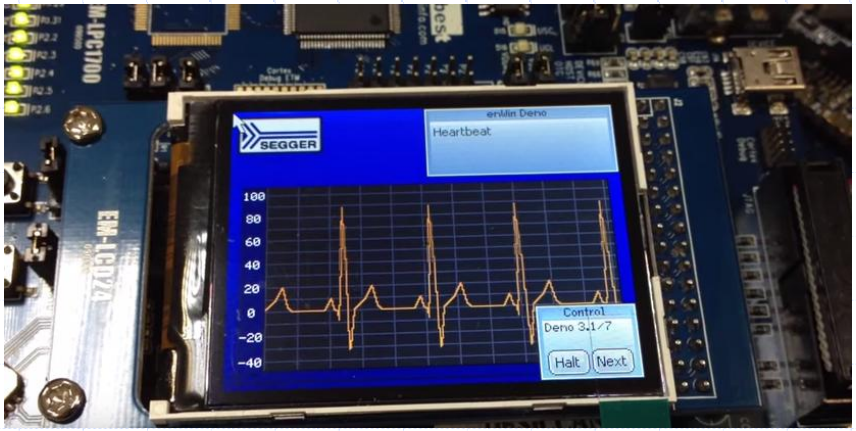
# Alternatives to CMSIS/RTX RTOS

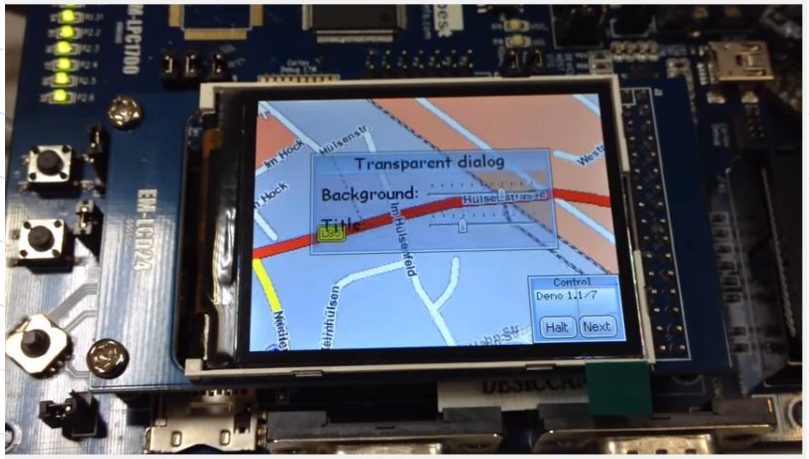
- **VxWorks** – proprietary and customizable RTOS
  - Works on a variety of uControllers/uProcessors
  - WindRiver kernel
  - Lightweight User interface – launch terminal, issue commands etc
- **FreeRTOS** – low power applications, compatible with a variety of uC/ uP

# emWin

Universal graphic software for embedded applications




# Final Project – Media Center

# Media Center – Final Project

- Upon launch – graphical interface appears with a menu of options, select with joystick
- Media Center at a minimum should provide the following functions:
  - A photo gallery
  - Mp3 player
  - Game(s)
  - Any other functionality you wish to implement

# Bonus Projects

- Bonus topic is selected, 1 per section:
  - Media Center with complex mp3 player
    - Use on-board USB drive or SD card so that the user may select from multiple mp3s
  - USB keyboard integration for gameplay and menu navigation
  - Port emWin to MCB1700. Create an embedded application (discuss with professor/TA)
  - Port FreeRTOS to MCB1700. Create an embedded application (discuss - -” - -)

# Bonus Projects

- Bonus topic is selected, 1 per section:
  - Use CAN protocol to establish communication between 2 boards, implementing a real-time application
  - Use ethernet protocol/stack to establish communication between 2 boards, implementing a real-time application
  - Integrate Nintendo Nunchuck to the media player to navigate through the menu and photo gallery (no joystick)
    - Requires I2C protocol knowledge