CEC Continuing Education Center

DesignNews

Introduction to Multicore RTOS-based Application Development

DAY 2 : A Quick Review of RTOS Fundamentals

Sponsored by

Digi-Key ELECTRONICS

informa markets

Webinar Logistics

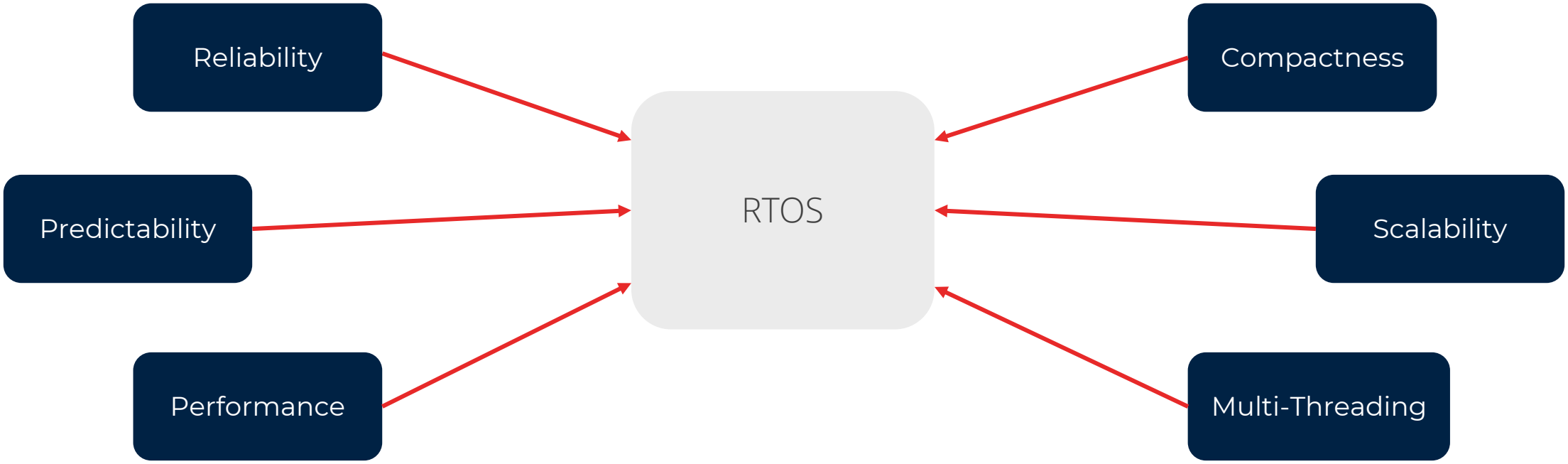
- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.
- Submit questions for the lecturer using the Q&A widget. They will follow-up after the lecture portion concludes.

Course Sessions

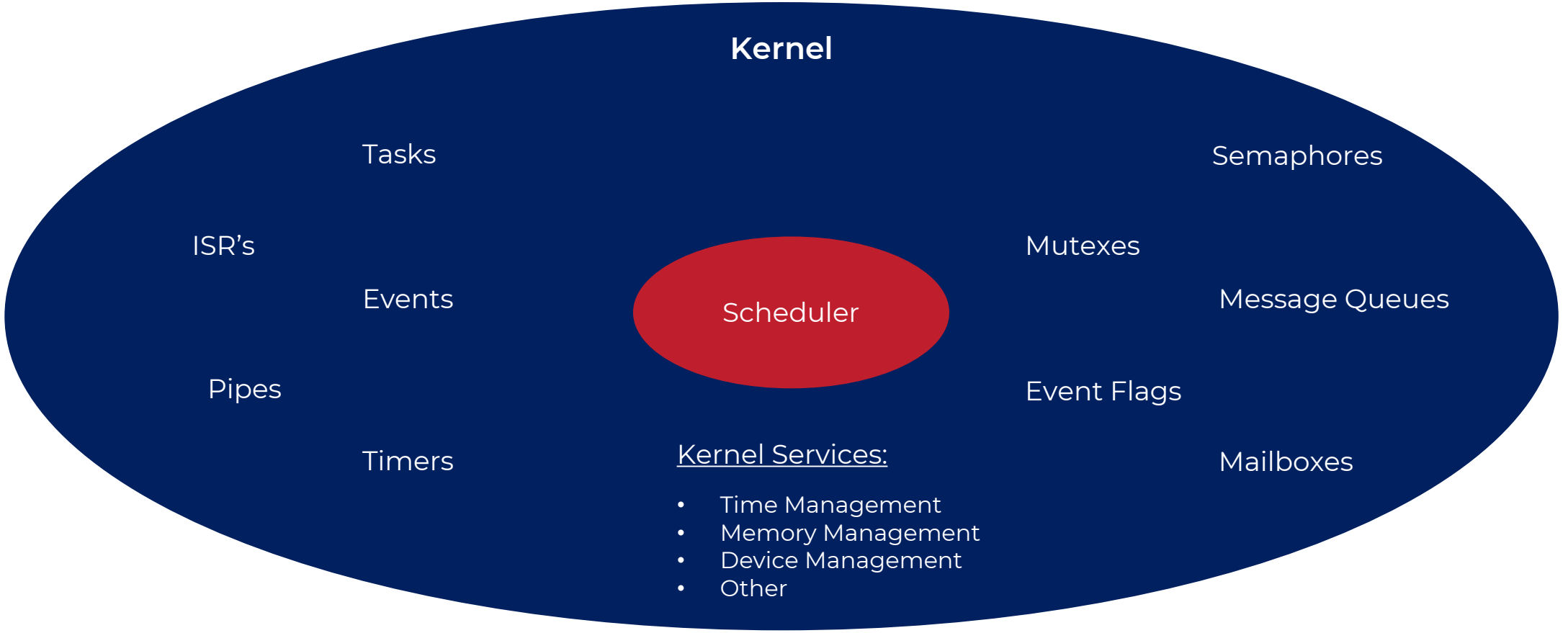
- Multicore Application Architecture Design
- A Quick Review of RTOS Fundamentals
- Digging into the Dual-Core STM32H7 MCU's
- Toolchain Setup for Dual Core MCU's
- Writing Multicore Microcontroller Applications

RTOS Characteristics

A **Real-Time Operating System (RTOS)** is an operating system designed to manage hardware resources of an embedded system with very precise timing and a high degree of reliability.



Real-time Operating Systems



Real-time Operating Systems

- Provide developers with a mechanism to reliably schedule tasks
- Create the illusion that tasks are running concurrently
- Include basic capabilities to synchronize tasks
- Mechanisms to protect
  - Shared resources
  - Critical sections
- Developers still need to mind their task timing!
- May be certified to ensure the kernel won't introduce bugs!

Tasks, Thread and Processes

Definition:

[1] A task is a concurrent and independent program that competes for execution time on a CPU.
[2] a semi-independent portion of the application that carries out a specific duty.

Definition:

[1] A thread is a semi-independent program segment that executes within a process.

Definition:

[1] A process is a collection of threads and associated memory that run in an independent memory location.

Task Anatomy in FreeRTOS

Task initialization data

```
void Task_LedBlink( void *pvParameters )
{
    const TickType_t xDelay = 500 / portTICK_PERIOD_MS;

    for(;;)
    {
        HAL_GPIO_TogglePin(GPIOB, LED2_Pin);

        vTaskDelay(xDelay);
    }
}
```

Semi-independent program that looks like main()

Task yields until event!

RTOS Applications

Semi-independent Programs

```
128 /*lint -save  -e970 Disable MISRA rule (6.3) checking. */
129 int main(void)
130 /*lint -restore Enable MISRA rule (6.3) checking. */
131 {
132   /* Write your local variable definition here */
133
134   /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
135   PE_low_level_init();
136   /*** End of Processor Expert internal initialization.                    ***/
137
138   /* Write your code here */
139   /* For example: for(;;) { } */
140   xTaskCreate(Led_GreenBlink,                /* Task Pointer */
141               (const char* const)"led_green",/* Task Name */
142               configMINIMAL_STACK_SIZE,      /* Stack Depth */
143               0,                             /* Parameters to pass to task*/
144               3,                             /* Task Priority */
145               0);                            /* Pass handle to created task */
146
147   xTaskCreate(Led_RedBlink,                  /* Task Pointer */
148               (const char* const)"led_red",  /* Task Name */
149               configMINIMAL_STACK_SIZE,      /* Stack Depth */
150               0,                             /* Parameters to pass to task*/
151               2,                             /* Task Priority */
152               0);                            /* Pass handle to created task */
153
154   xTaskCreate(Led_BlueBlink,                 /* Task Pointer */
155               (const char* const)"led_blue", /* Task Name */
156               configMINIMAL_STACK_SIZE,      /* Stack Depth */
157               0,                             /* Parameters to pass to task*/
158               1,                             /* Task Priority */
159               0);                            /* Pass handle to created task */
```

```
64 void Led_BlueBlink(void *pvParameters)
65 {
66     const TickType_t xDelay = 500 / portTICK_PERIOD_MS;
67     uint32_t BlueDelay = 0;
68     const uint32_t TargetCount = 160000;
69
70     for(;;)
71     {
72         LED_Blue_On();
73         Delay_Nonsense(&BlueDelay, &TargetCount);
74         vTaskDelay(xDelay);
75         LED_Blue_Off();
76         Delay_Nonsense(&BlueDelay, &TargetCount);
77         vTaskDelay(xDelay);
78     }
79 }
```

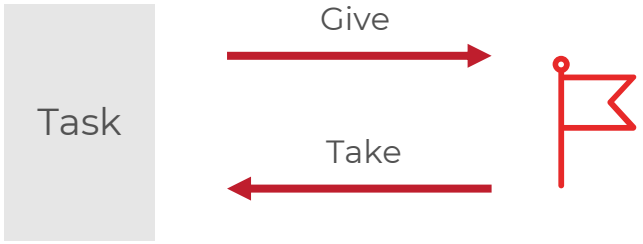
```
81 void Led_RedBlink( void *pvParameters )
82 {
83     const TickType_t xDelay = 100 / portTICK_PERIOD_MS;
84     uint32_t RedDelay = 0;
85     const uint32_t TargetCount = 160000;
86
87     for( ;; )
88     {
89         LED_Red_On();
90         Delay_Nonsense(&RedDelay, &TargetCount);
91         vTaskDelay( xDelay );
92         LED_Red_Off();
93         Delay_Nonsense(&RedDelay, &TargetCount);
94         vTaskDelay( xDelay );
95     }
96 }
```

Which of the following is the definition of a process?

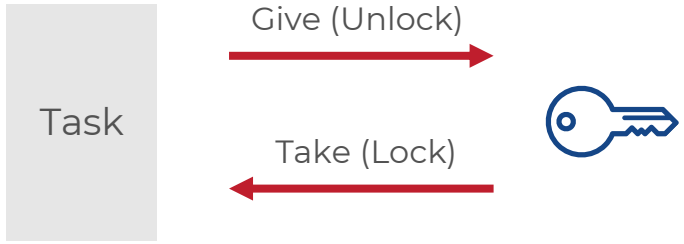
- a concurrent and independent program that competes for execution time on a CPU.
- a semi-independent program segment
- a collection of threads and associated memory that run in an independent memory location

Application Synchronization and Notification

Semaphores (Sync and Notify)

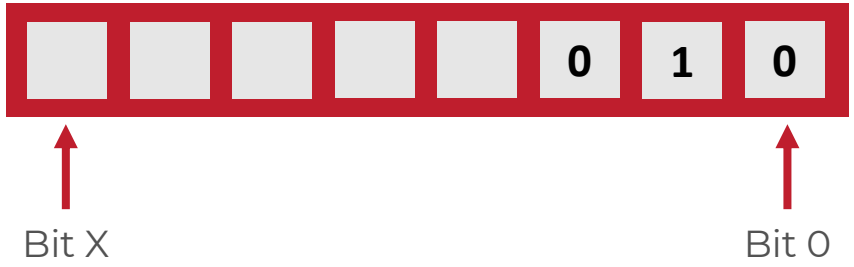


Mutexes (Mutual Exclusion)



Message Queues (Communication)

Event Flags (Synchronization)



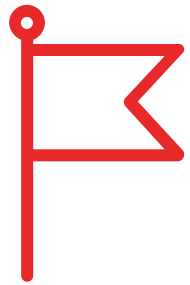
Semaphores

A semaphore is used to synchronize application behavior between

- An ISR and a task
- One task to another task
- Occasionally are used to protect a shared resource but this is not their primary purpose

Semaphores use flags or tokens to count between 0 and a maximum set value.

- Binary Semaphores
- Counting Semaphores



Mutexes

A mutex is used to gain access to a shared resource such as

- A memory location
- A common peripheral like a UART

Mutexes have the concept of providing a task ownership over the shared resource.

Mutexes have a property known as priority inheritance which protects an application from priority inversions.

Message Queues

A message queue is a buffer-like structure that can be used for

- Synchronization
- Passing data between tasks

Can be configured for

- First In First Out (FIFO)
- Last in First Out (LIFO)

Event Flags

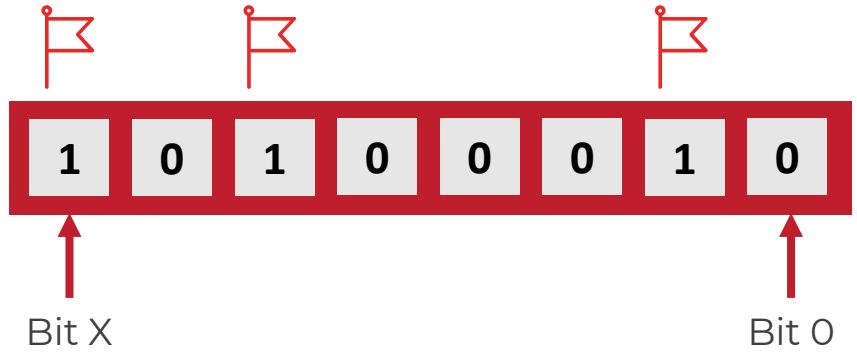
Event flags (groups) are a group of individual bits that are used for task

- Synchronization
- Notification

Each bit is considered a flag that notifies when an event has occurred. They can be triggered by

- Interrupts
- Tasks

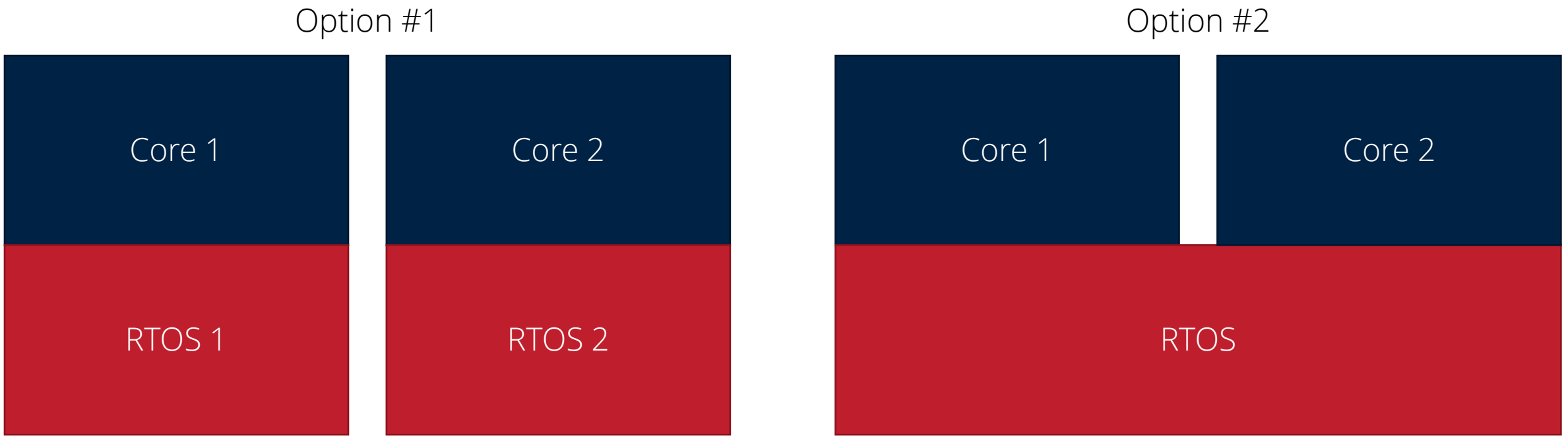
May be 8, 16, or 32 bits wide



Which of the following is used to provide mutually exclusive access to a resource?

- Semaphore
- Mutex
- Message Queue
- Event Flags

Multicore RTOS Configurations



Which RTOS configuration do you think is the most common?

- Same RTOS for each core
- Different RTOS for each core
- One RTOS for each core
- Other

Thank you for attending

Please consider the resources below:

- www.beningo.com
  - Blog, White Papers, Courses
  - Embedded Bytes Newsletter
    - http://bit.ly/1BAHYXm

From www.beningo.com under

- Blog > CEC – Introduction to Multicore RTOS-based Application Development

CEC Continuing Education Center

DesignNews

Thank You

Sponsored by

Digi-Key ELECTRONICS

informa markets