Embedded System Design Techniques™

Connecting Edge Devices to the IoT using Amazon FreeRTOS

Class 4: Amazon FreeRTOS Behind the Scenes

March 22nd, 2018

Jacob Beningo

Presented by:

CEC CONTINUING EDUCATION CENTER

Digi-Key ELECTRONICS

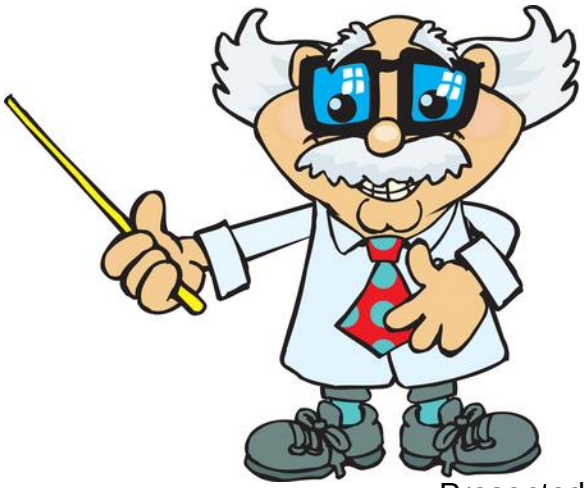
Course Overview

Topics:

- Introduction to Amazon FreeRTOS
- Amazon Web Services Fundamentals
- Setting up and Configuring Amazon FreeRTOS
- **Amazon FreeRTOS Behind the Scenes**
- Creating your own a:FreeRTOS Application

Session Overview

- The FreeRTOS Echo Demo
- Tracing the application
- Walking through the code
- Problems you will encounter



Presented by:

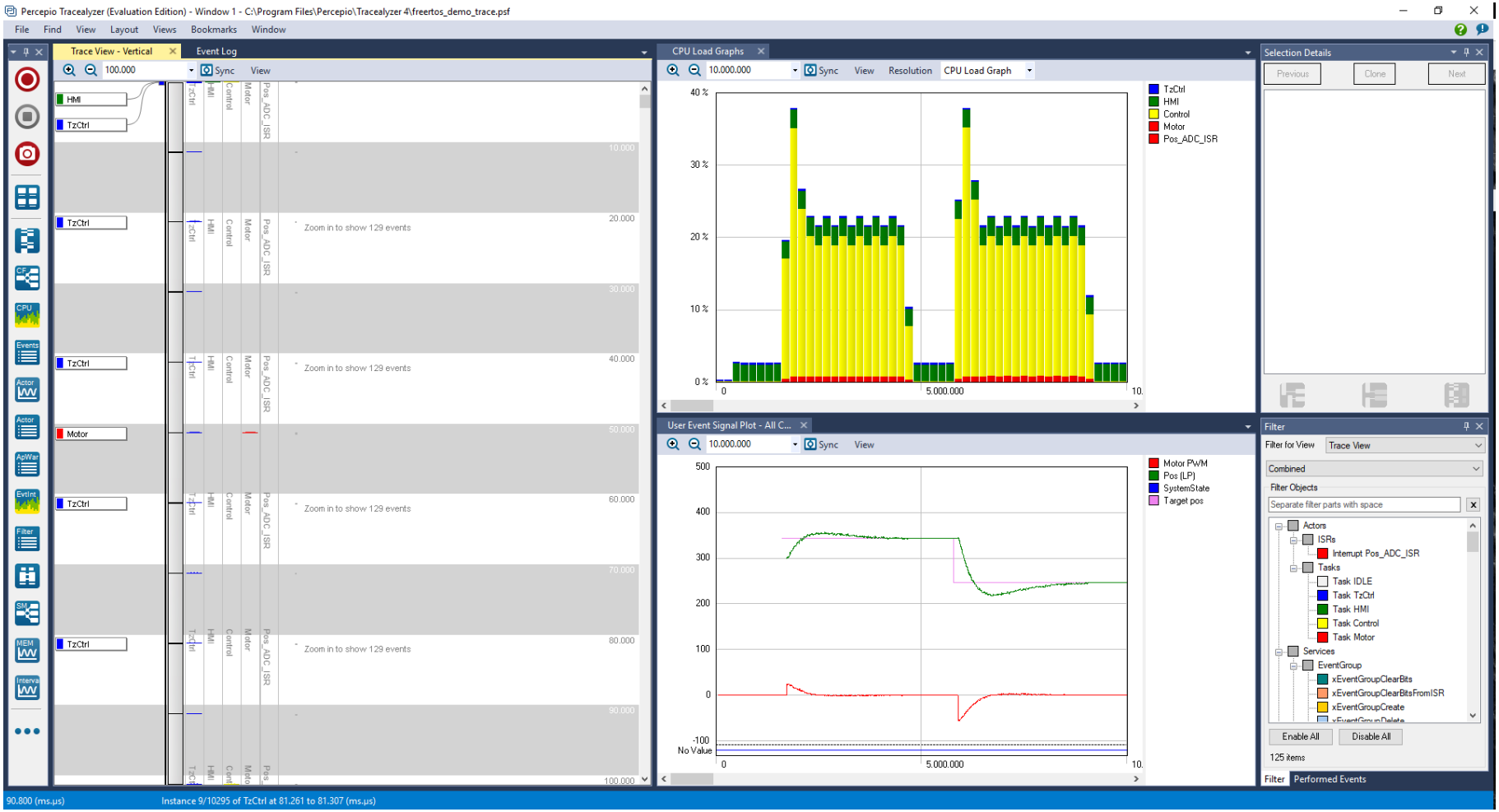
A:FreeRTOS Echo Demo

What can go wrong?

Code is provided as an example not a production intent solution

- Issues programming the device
- Communication issues
- Incorrect directions
- Holes in the documentation
- Unclear process on how to adapt the example
- Steps to productionize the example

Trace the Application



Trace the Application

Count: Number of instances of an actor accounted for in this report.

CPU Usage: The amount of cpu usage of an actor relative to cpu capacity.

Execution Time: Time spent executing an instance.

Response Time: Time from start of an instance until its completion.

Fragmentation: The number of fragments of an actor instance (due to context-switching or interrupts).

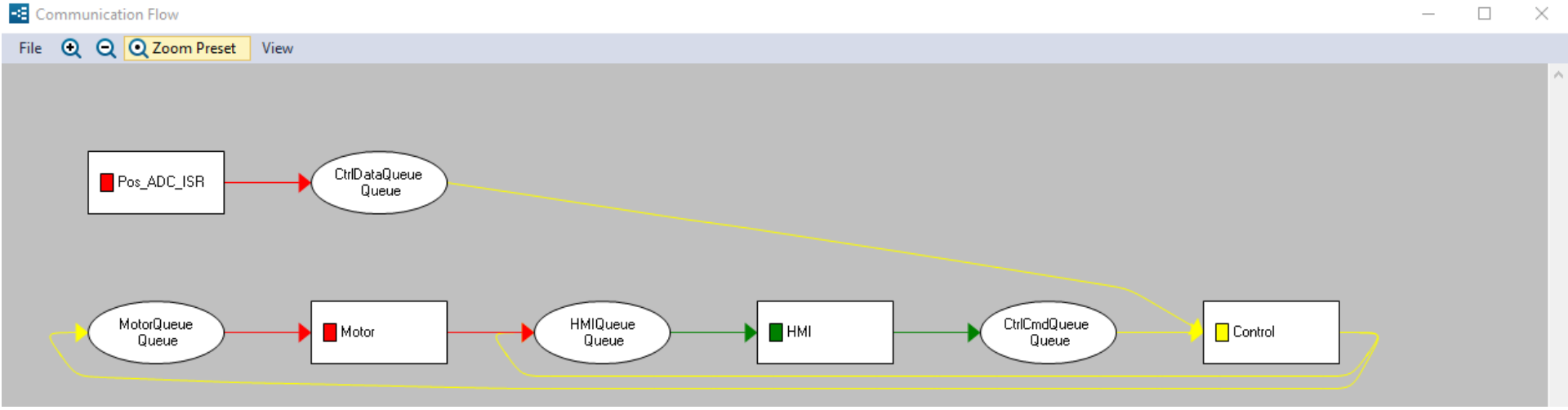
Separation: The time between the end of an instance and the start of the next instance.

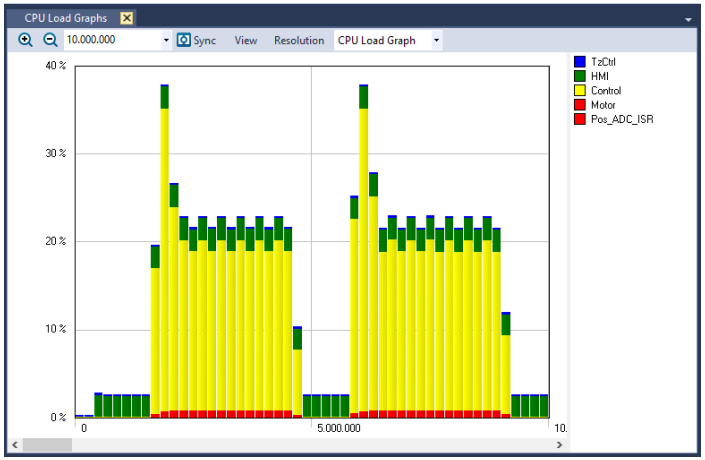
Periodicity: The time between the start of an instance and the start of the next instance.

Actor	Count	CPU Usage	Execution Time			Response Time			Fragmentation			Separation (Ready)		
			Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
IDLE	1	84.561	1:30.745.928	1:30.745.928	1:30.745.928	1:47.316.856	1:47.316.856	1:47.316.856	51434	51434	51434	N/A	N/A	N/A
TzCtrl	10295	0.238	25	25	25	25	637	490.721	1	1	1	9.169	9.787	9.972
HMI	1062	3.315	24	3.350	495.381	24	3.846	496.115	1	2.115	11	3.885	97.283	499.889
Control	23432	10.725	62	491	2.651	63	533	2.764	1	1.334	3	900	4.047	99.914
Motor	7236	0.457	64	68	93	75	99	138	1	1	1	6.459	14.726	99.908
Pos_ADC_ISR	26813	0.705	19	28	30	19	28	30	1	1	1	1.943	3.847	3.800.980

Separation (Exec)			Periodicity (Ready)			Periodicity (Exec)		
Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
9.288	10.400	500.650	9.889	10.424	500.000	9.313	10.424	500.675
3.907	97.527	499.999	99.991	101.131	500.000	97.246	101.131	500.023
918	4.074	99.943	1.028	4.580	101.971	1.017	4.580	101.974
6.471	14.758	99.929	6.597	14.826	100.000	6.542	14.826	100.001
1.943	3.847	3.800.980	1.973	3.875	3.801.000	1.973	3.875	3.801.000

Trace the Application





Presented by:

The main function

```
78 /**
79  * @brief Performs board and logging initializations, then starts the OS.
80  *
81  * Functions that require the OS to be running
82  * are handled in vApplicationDaemonTaskStartupHook().
83  *
84  * @sa vApplicationDaemonTaskStartupHook()
85  *
86  * @return This function should not return.
87  */
88 int main( void )
89 {
90     /* Call board init functions. */
91     Board_initGeneral();
92
93     /* Start logging task. */
94     xLoggingTaskInitialize( democonfigTASKSTACKSIZE,
95                             tskIDLE_PRIORITY,
96                             mainLOGGING_MESSAGE_QUEUE_LENGTH );
97
98 //    vTraceEnable(TRC_START_AWAIT_HOST);
99
100    /* Start the FreeRTOS scheduler. */
101    vTaskStartScheduler();
102
103    return( 0 );
104 }
```

Logging Task

```
75 /*-----------------------------------------------------------*/
76
77 /*
78  * The queue used to pass pointers to log messages from the task that created
79  * the message to the task that will performs the output.
80  */
81 static QueueHandle_t xQueue = NULL;
82
83 /*-----------------------------------------------------------*/
84
85 BaseType_t xLoggingTaskInitialize( uint16_t usStackSize, UBaseType_t uxPriority, UBaseType_t uxQueueLength )
86 {
87 BaseType_t xReturn = pdFAIL;
88
89     /* Ensure the logging task has not been created already. */
90     if( xQueue == NULL )
91     {
92         /* Create the queue used to pass pointers to strings to the logging task. */
93         xQueue = xQueueCreate( uxQueueLength, sizeof( char ** ) );
94
95         if( xQueue != NULL )
96         {
97             if( xTaskCreate( prvLoggingTask, "Logging", usStackSize, NULL, uxPriority, NULL ) == pdPASS )
98             {
99                 xReturn = pdPASS;
100            }
101            else
102            {
103                /* Could not create the task, so delete the queue again. */
104                vQueueDelete( xQueue );
105            }
106        }
107    }
108
109    return xReturn;
110 }
```

Logging Task

```
113 static void prvLoggingTask( void *pvParameters )
114 {
115 char *pcReceivedString;
116
117     for( ;; )
118     {
119         /* Block to wait for the next string to print. */
120         if( xQueueReceive( xQueue, &pcReceivedString, portMAX_DELAY ) == pdPASS )
121         {
122             configPRINT_STRING( pcReceivedString );
123             vPortFree( ( void * ) pcReceivedString );
124         }
125     }
126 }
```

But was is the meat?

main.c includes

- vApplicationDaemonTaskStartupHook

Daemon Task Startup Hook

The RTOS daemon task is the same as the Timer Service Task. Sometimes it is referred to as the daemon task because the task is now used for more than just servicing timers.

If configUSE_DAEMON_TASK_STARTUP_HOOK is set to 1 in FreeRTOSConfig.h then the Daemon Task Startup Hook will be called as soon as the Daemon Task starts executing for the first time. This is useful if the application includes initialisation code that would benefit from executing after the scheduler has been started, which allows the initialisation code to make use of the RTOS functionality.

If configUSE_DAEMON_TASK_STARTUP_HOOK is set to 1 then the application writer must provide an implementation of the Daemon Task startup hook function with the following name an prototype.

```
void vApplicationDaemonTaskStartupHook( void );
```

Daemon Task Start-up Hook

FreeRTOSConfig.h

```
49 *-----------------------------------------------------------*/
50
51 #define configCPU_CLOCK_HZ                    ( ( unsigned long ) 80000000 )
52
53 #define configUSE_DAEMON_TASK_STARTUP_HOOK    1
```

```
114 void vApplicationDaemonTaskStartupHook( void )
115 {
116     UART_Handle xtUartHndl;
117     WIFINetworkParams_t xNetworkParams;
118
119     Board_initGPIO();
120     Board_initSPI();
121
122     /* Configure the UART. */
123     xtUartHndl = InitTerm();
124     UART_control( xtUartHndl, UART_CMD_RXDISABLE, NULL );
125
126     /* Initialize WiFi module, start simple link device. */
127     WIFI_On();
```

Daemon Task Start-up Hook

```
129     /* A simple example to demonstrate key and certificate provisioning in
130      * flash using PKCS#11 interface. This should be replaced
131      * by production ready key provisioning mechanism. This function must be called after
132      * initializing the TI File System using WIFI_On. */
133     vDevModeKeyProvisioning();
134
135     /* Initialize the AWS library system. */
136     BaseType_t xResult = SYSTEM_Init();
137     configASSERT( xResult == pdPASS );
138
139     /* Initialize Network params. */
140     xNetworkParams.pcSSID = clientcredentialWIFI_SSID;
141     xNetworkParams.ucSSIDLength = sizeof( clientcredentialWIFI_SSID );
142     xNetworkParams.pcPassword = clientcredentialWIFI_PASSWORD;
143     xNetworkParams.ucPasswordLength = sizeof( clientcredentialWIFI_PASSWORD );
144     xNetworkParams.xSecurity = clientcredentialWIFI_SECURITY;
145
146     /* Connect to WIFI. */
147     WIFI_ConnectAP( &xNetworkParams );
148
149     DEMO_RUNNER_RunDemos();
150 }
```

DEMO_RUNNER_RunDemos

```
36 void DEMO_RUNNER_RunDemos( void )
37 {
38     vStartMQTTEchoDemo();
39 }
```

```
490 void vStartMQTTEchoDemo( void )
491 {
492     configPRINTF( ( "Creating MQTT Echo Task...\r\n" ) );
493
494     /* Create the message buffer used to pass strings from the MQTT callback
495      * function to the task that echoes the strings back to the broker.  The
496      * message buffer will only ever have to hold one message as messages are only
497      * published every 5 seconds.  The message buffer requires that there is space
498      * for the message length, which is held in a size_t variable. */
499     xEchoMessageBuffer = xMessageBufferCreate( ( size_t ) echoMAX_DATA_LENGTH + sizeof( size_t ) );
500     configASSERT( xEchoMessageBuffer );
501
502     /* Create the task that publishes messages to the MQTT broker every five
503      * seconds.  This task, in turn, creates the task that echoes data received
504      * from the broker back to the broker. */
505     ( void ) xTaskCreate( prvMQTTConnectAndPublishTask,        /* The function that implements the demo task. */
506                           "MQTTEcho",                          /* The name to assign to the task being created. */
507                           democonfigMQTT_ECHO_TASK_STACK_SIZE, /* The size, in WORDS (not bytes), of the stack to allocate for the task being created. */
508                           NULL,                                /* The task parameter is not being used. */
509                           democonfigMQTT_ECHO_TASK_PRIORITY,   /* The priority at which the task being created will run. */
510                           NULL );                              /* Not storing the task's handle. */
511 }
```

prvMQTTConnectAndPublishTask

```
422 static void prvMQTTConnectAndPublishTask( void * pvParameters )
423 {
424     BaseType_t x, xReturned;
425     const TickType_t xFiveSeconds = pdMS_TO_TICKS( 5000UL );
426     const BaseType_t xIterationsInAMinute = 60 / 5;
427     TaskHandle_t xEchoingTask = NULL;
428
429     /* Avoid compiler warnings about unused parameters. */
430     ( void ) pvParameters;
431
432     /* Create the MQTT client object and connect it to the MQTT broker. */
433     xReturned = prvCreateClientAndConnectToBroker();
434
435     if( xReturned == pdPASS )
436     {
437         /* Create the task that echoes data received in the callback back to the
438          * MQTT broker. */
439         xReturned = xTaskCreate( prvMessageEchoingTask,              /* The function that implements the task. */
440                                  "Echoing",                          /* Human readable name for the task. */
441                                  democonfigMQTT_ECHO_TASK_STACK_SIZE, /* Size of the stack to allocate for the task, in words not bytes! */
442                                  NULL,                               /* The task parameter is not used. */
443                                  tskIDLE_PRIORITY,                   /* Runs at the lowest priority. */
444                                  &( xEchoingTask ) );                /* The handle is stored so the created task can be deleted again at the end of the demo. */
445
446         if( xReturned != pdPASS )
447         {
448             /* The task could not be created because there was insufficient FreeRTOS
449              * heap available to create the task's data structures and/or stack. */
450             configPRINTF( ( "MQTT echoing task could not be created - out of heap space?\r\n" ) );
451         }
452     }
453     else
454     {
455         configPRINTF( ( "MQTT echo test could not connect to broker.\r\n" ) );
456     }
```

prvMQTTConnectAndPublishTask

```
458     if( xReturned == pdPASS )
459     {
460         configPRINTF( ( "MQTT echo test echoing task created.\r\n" ) );
461
462         /* Subscribe to the echo topic. */
463         xReturned = prvSubscribe();
464     }
465
466     if( xReturned == pdPASS )
467     {
468         /* MQTT client is now connected to a broker.  Publish a message
469          * every five seconds until a minute has elapsed. */
470         for( x = 0; x < xIterationsInAMinute; x++ )
471         {
472             prvPublishNextMessage( x );
473
474             /* Five seconds delay between publishes. */
475             vTaskDelay( xFiveSeconds );
476         }
477     }
478
479     /* Disconnect the client. */
480     ( void ) MQTT_AGENT_Disconnect( xMQTTHandle, democonfigMQTT_TIMEOUT );
481
482     /* End the demo by deleting all created resources. */
483     configPRINTF( ( "MQTT echo demo finished.\r\n" ) );
484     vMessageBufferDelete( xEchoMessageBuffer );
485     vTaskDelete( xEchoingTask );
486     vTaskDelete( NULL ); /* Delete this task. */
487 }
```

Additional Resources

- Download Course Material for
  - C/C++ Doxygen Templates
  - Example source code
  - Blog
  - YouTube Videos
- Embedded Bytes Newsletter
  - http://bit.ly/1BAHYXm

From www.beningo.com under

- Blog > CEC – Connecting Edge Devices to the IoT using Amazon FreeRTOS