ECE3849
D-Term 2021

Real Time Embedded Systems

Module 4 Part 2

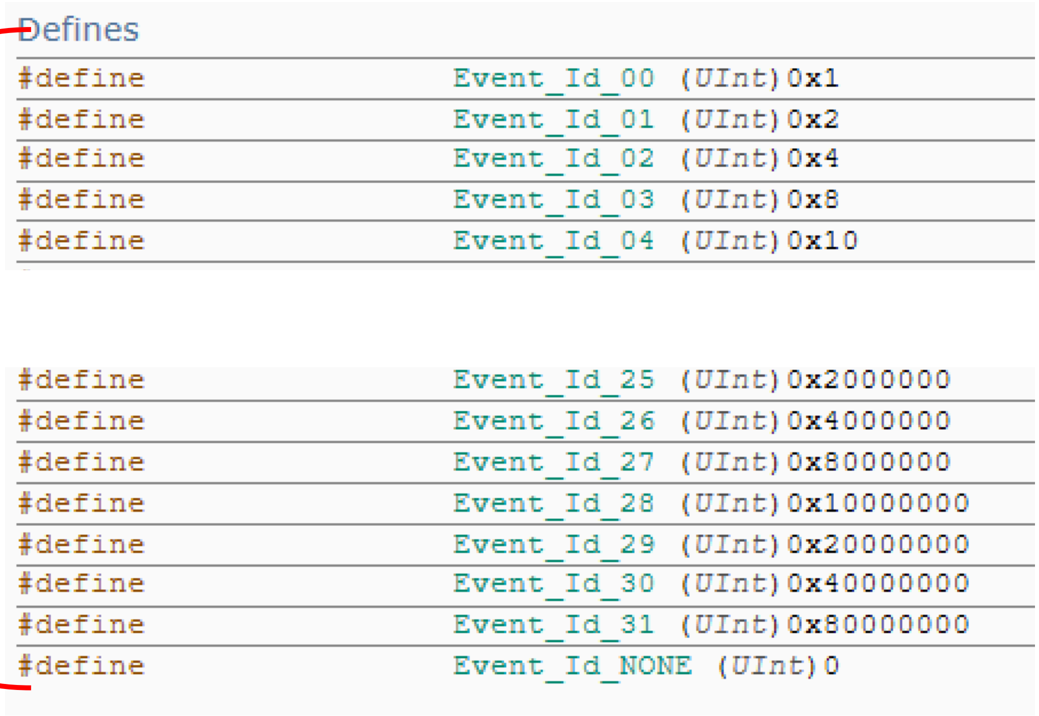
Module 4 Part 2 Overview

- Event Objects.
- Clock Module services.
- Example Using Events and Clock Objects.

Event Objects

- The event objects allows a task to wait for multiple conditions, “events”, before signaling.
  - Events are similar to simultaneously waiting on multiple semaphores.
  - Event_post() command signals that an event or set of events have occurred.
  - Event_pend() command waits for the specified events to be signaled.
- Unlike semaphores, only a single task may pend on a specific event instance in the TI-RTOS.

- Events are typically grouped in sets of 32 or 16.
  - For TI-RTOS a single event instance can manage up to 32 events.
  - Each event in the instance has a designated bit and behaves like a binary semaphore.
    - If an event is posted, its corresponding bit will be set.
- A task designates which event it wishes to wait for or signal using a bit mask.

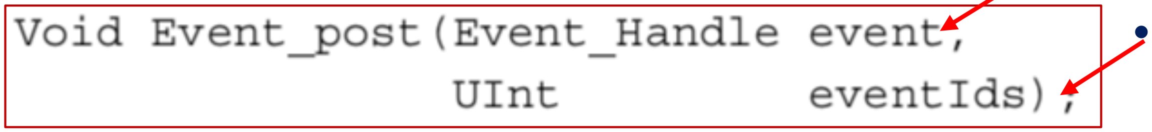


Defines

```
#define        Event_Id_00  (UInt)0x1
#define        Event_Id_01  (UInt)0x2
#define        Event_Id_02  (UInt)0x4
#define        Event_Id_03  (UInt)0x8
#define        Event_Id_04  (UInt)0x10


#define        Event_Id_25  (UInt)0x2000000
#define        Event_Id_26  (UInt)0x4000000
#define        Event_Id_27  (UInt)0x8000000
#define        Event_Id_28  (UInt)0x10000000
#define        Event_Id_29  (UInt)0x20000000
#define        Event_Id_30  (UInt)0x40000000
#define        Event_Id_31  (UInt)0x80000000
#define        Event_Id_NONE  (UInt)0
```

Event_post() function

- Event_post() prototype

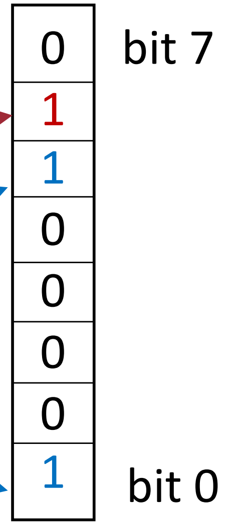


```
Void Event_post(Event_Handle event,
                UInt         eventIds);
```

- Event instance name.
- Which events to be signaled.
  - A post can signal one event or multiple events.

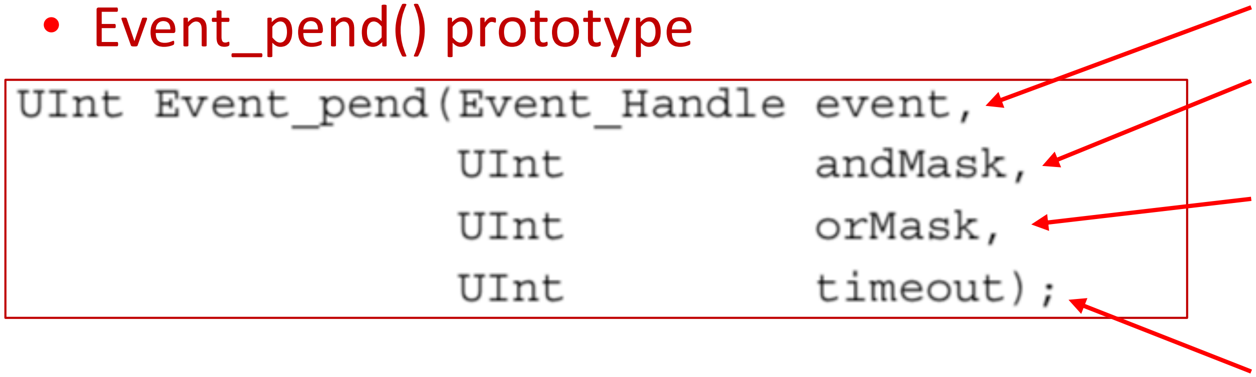
- TI-RTOS allows other objects such as semaphores and mailboxes to automatically post a specified event in a specified event object.
  - Semaphores can post events on a Semaphore_post().
  - Mailboxes can post events on a Mailbox_post() or Mailbox_pend().
- Hwi, Swi and tasks can also post to an event instance.
- When an event is posted a 1 is placed in its bit mask location, leaving other event status unchanged.

```
Event_post(myEvent, Event_Id_06);
Event_post(myEvent, (Event_Id_00 | Event_Id_05));
```



Event_pend() Function

- A task can wait for multiple events to be signaled before unblocking.
  - AND Operation: It can wait for all events in its andMask bit mask to be signaled before unblocking.
  - OR Operation: It can wait for any one event in its orMask bit mask to be signaled before unblocking.
- Event_pend() prototype



```
UInt Event_pend(Event_Handle event,
                UInt         andMask,
                UInt         orMask,
                UInt         timeout);
```

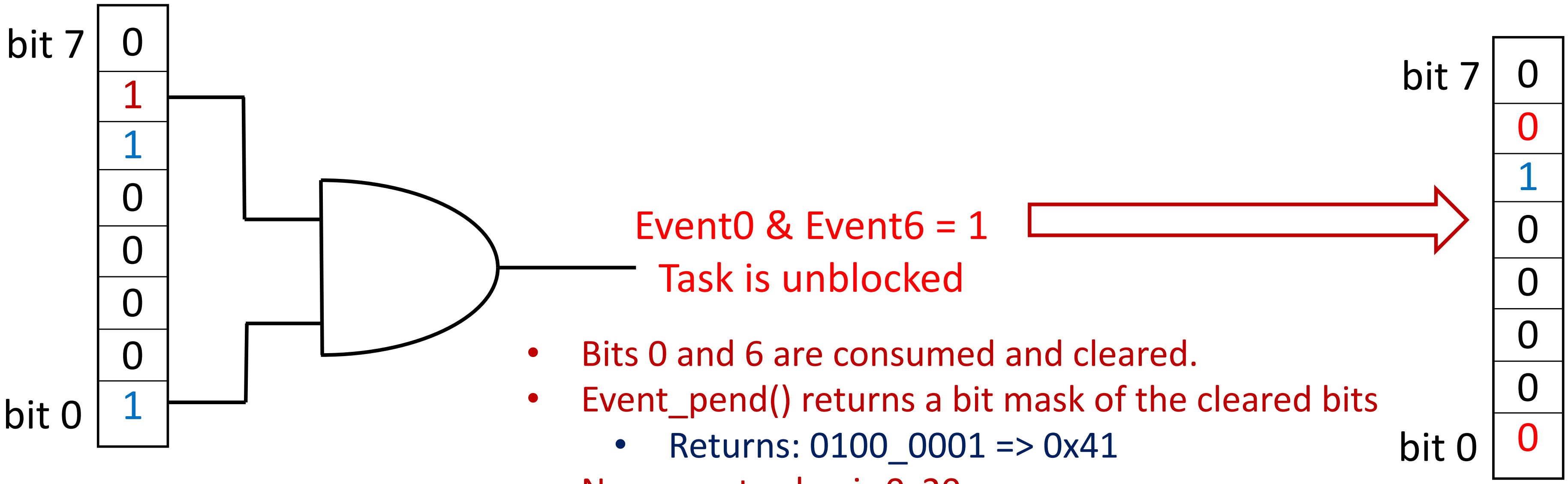
- Event instance name.
- All events in this mask must be signaled to unblock.
- Any one of the event in this mask can be signaled to unblock.
- How long to pend before timing out.

- Tasks may pend on events using any timeout value.
  - BIOS_WAIT_FOREVER will pend until the event conditions are met. This is the recommended setting for tasks.
- Hwi and Swi may not block on events.
  - They may use the Event_pend() command using the BIOS_NO_WAIT timeout value in a polling mode.

Event_pend() Return Value

- An Event_pend() is unblocked when either its andMask condition OR its orMask condition is met.
  - If the AND mask was satisfied, all events in the andMask bit mask are cleared.
  - If the OR mask was satisfied, all events in the orMask bit mask are cleared.
  - If one of the masks is not used Event_Id_NONE can be entered for its argument.
- Event_pend() returns a bit mask of all active events that were cleared (consumed).
- Example:
  - Requires Event 0 AND Event 6 be signaled.

```
Event_pend(myEvent, (Event_Id_00 | Event_Id_06), Event_Id_NONE,
BIOS_WAIT_FOREVER);
```



Event0 & Event6 = 1
Task is unblocked

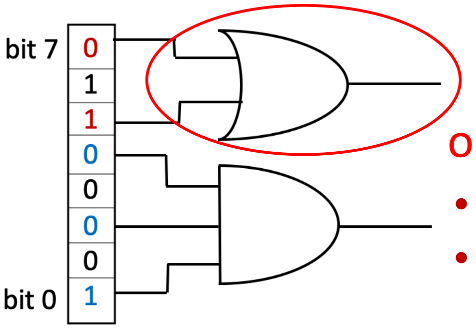
- Bits 0 and 6 are consumed and cleared.
- Event_pend() returns a bit mask of the cleared bits
  - Returns: 0100_0001 => 0x41
- New event value is 0x20

Event_pend() Example

- Example: Two conditions can unblock this event.
  - andMask: Event0 and Event2 and Event4
  - orMask: Event5 or Event 7

```
myResult = Event_pend(myEvent,
             (Event_Id_00 | Event_Id_02 | Event_Id_04),
             (Event_Id_05 | Event_Id_07), BIOS_WAIT_FOREVER);
```

Case 1



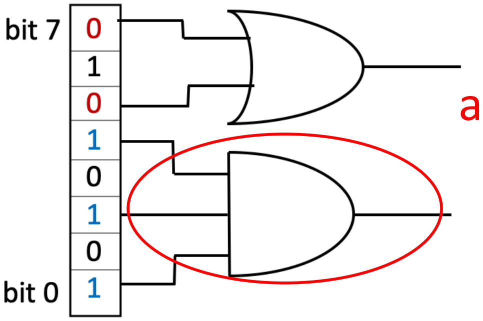
Event5 | Event7 = 1

orMask satisfied, task is unblocked

- Bit5 is consumed and cleared.
- Event_pend() returns a bit mask of the cleared bits
  - Returns: 0010_0000 => 0x20
- New event value is 0x41

bit 7
0
1
0
0
0
0
0
1
bit 0

Case 2



Event0 & Event2 & Event4 = 1

andMask satisfied, task is unblocked

- Bit0, 2 and 4 are consumed and cleared.
- Event_pend() returns a bit mask of the cleared bits
  - Returns: 0001_0101 => 0x15
- New event value is 0x40

bit 7
0
1
0
0
0
0
0
0
bit 0

Clock Module: Software Timers

- TI-RTOS Clock Module is a heartbeat timer or system tick timer that periodically interrupts and runs certain RTOS services.
  - It configures one of the hardware timers to perform this function.
  - One hardware timer to be shared across multiple clock services.
- Timer services run either in the timer ISR (Hwi) or in a high-priority software interrupt (Swi).
- Timer period is user specified
  - Shorter period gives more accurate timing but higher CPU load.
- Services provided in the TI_RTOS
  - Timeouts
    - Counts a certain number of ticks until threshold is reached.
  - Sleep
    - Task sleeps for a certain number of system clock ticks, similar to a pending state.
  - Calls a function periodically
    - Similar to int_latency example except using the clock module.
  - Call a function once after a specified delay (one shot).

Clock Module: Timeout Functions

- Most RTOS functions that can block have a timeout delay argument.
    - This argument is usually set to BIOS_WAIT_FOREVER for tasks or BIOS_NO_WAIT for interrupts.
- Functions called in tasks, can have a finite timeout value.
    - The Timeout argument is in clock ticks.
    - The clock tick period is configurable in the RTOS in units of usecs.
- Using finite timeouts is not recommended for normal operation.
    - Functions that use timeouts return false if a timeout occurs.
    - Timeout recovery becomes complicated.
        - It can lead to shared data problems as the critical section may still be locked.
- If handled properly timeouts can be helpful in
    - Resolving deadlock conditions.
    - Improving reliability by providing deterministic execution times.
    - In debugging to report if something is taking longer than expected.

Clock Module: Sleep

- Task_sleep()

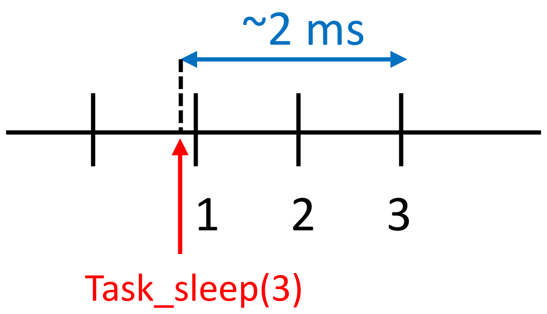
```
Void Task_sleep(UInt nticks);   // TI-RTOS sleep function
```

  - Input argument is in number of system clock ticks.
- A task calling the sleep function is blocked for a specified number of system ticks.
  - It is better than busy-wait delays executing in while loops, as it does not take CPU time.
  - It allows other tasks to execute while blocked.
- The timing accuracy depends on the clock tick period.
  - An clock tick of 1 msec will provide only 1 msec of accuracy.
  - A clock tick of 100 usec will provide 100 usec of accuracy.

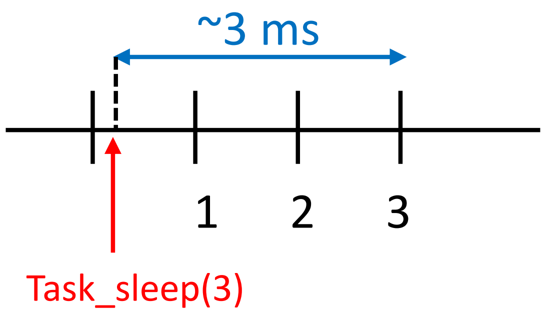
Clock Module: Sleep Accuracy

- Accuracy example
  - Task_sleep(3) with clock tick period of 1 msec.
  - Length of sleep depends on where function call happens in the clock tick period.

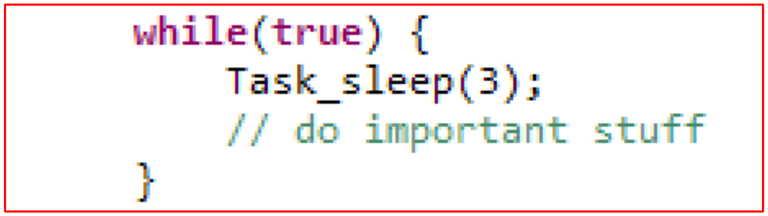
If a sleep occurs right before tick only sleeps for a little more than 2 ms.



If a sleep occurs right after tick only sleeps for a little less than 3 ms.

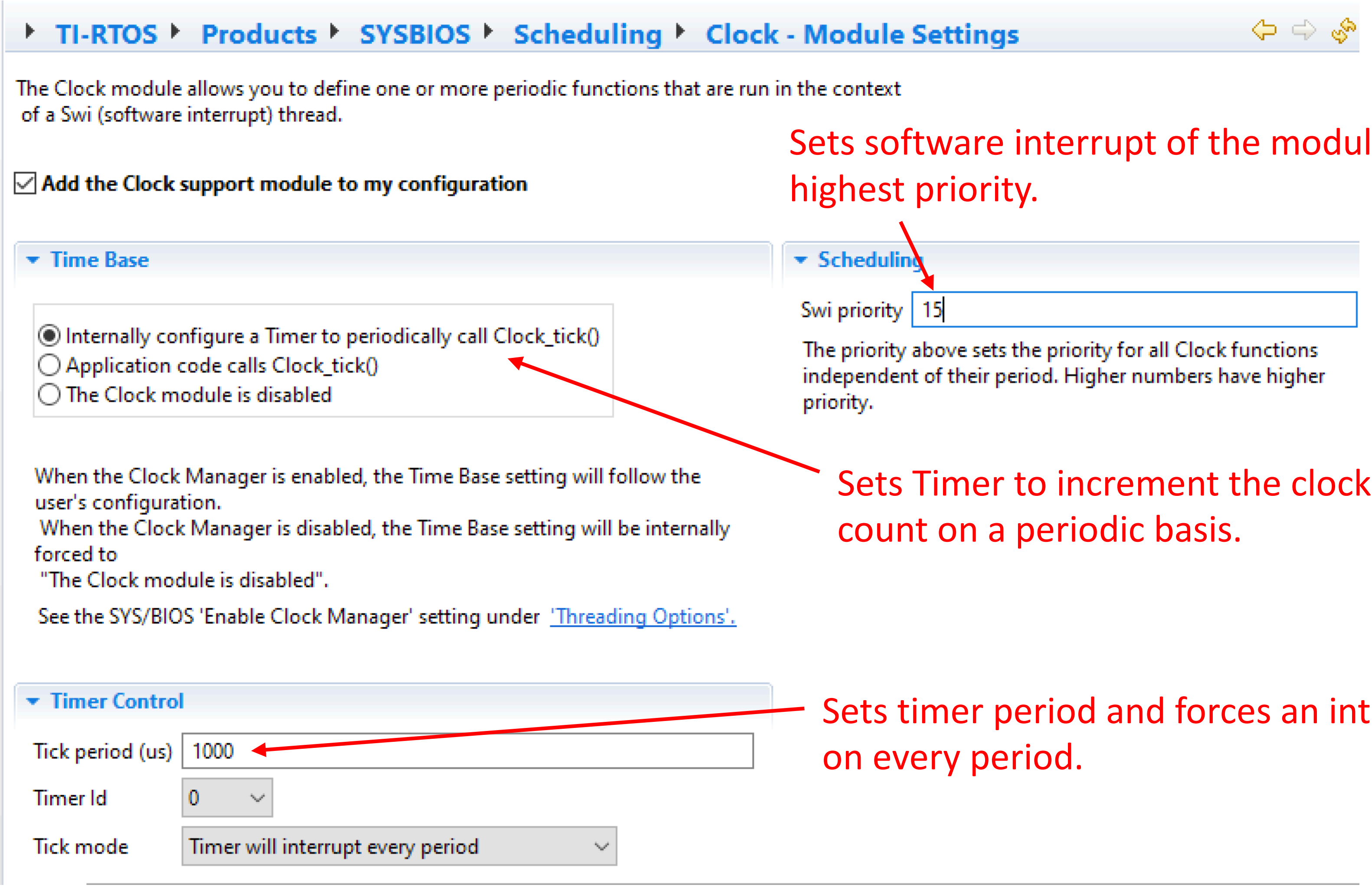


- If task_sleep in using in a while loop to trigger an event periodically the period accuracy will depend on,
  - The accuracy of the sleep.
  - The variability in execution time.
  - The variability in response time due to preemptions.



```
while(true) {
    Task_sleep(3);
    // do important stuff
}
```

Clock Module Configuration



▸ TI-RTOS ▸ Products ▸ SYSBIOS ▸ Scheduling ▸ Clock - Module Settings

The Clock module allows you to define one or more periodic functions that are run in the context of a Swi (software interrupt) thread.

☑ Add the Clock support module to my configuration

Time Base

● Internally configure a Timer to periodically call Clock_tick()
○ Application code calls Clock_tick()
○ The Clock module is disabled

When the Clock Manager is enabled, the Time Base setting will follow the user's configuration.
When the Clock Manager is disabled, the Time Base setting will be internally forced to
"The Clock module is disabled".

See the SYS/BIOS 'Enable Clock Manager' setting under 'Threading Options'.

Scheduling

Swi priority: 15

The priority above sets the priority for all Clock functions independent of their period. Higher numbers have higher priority.

Timer Control

Tick period (us)	1000
Timer Id	0
Tick mode	Timer will interrupt every period

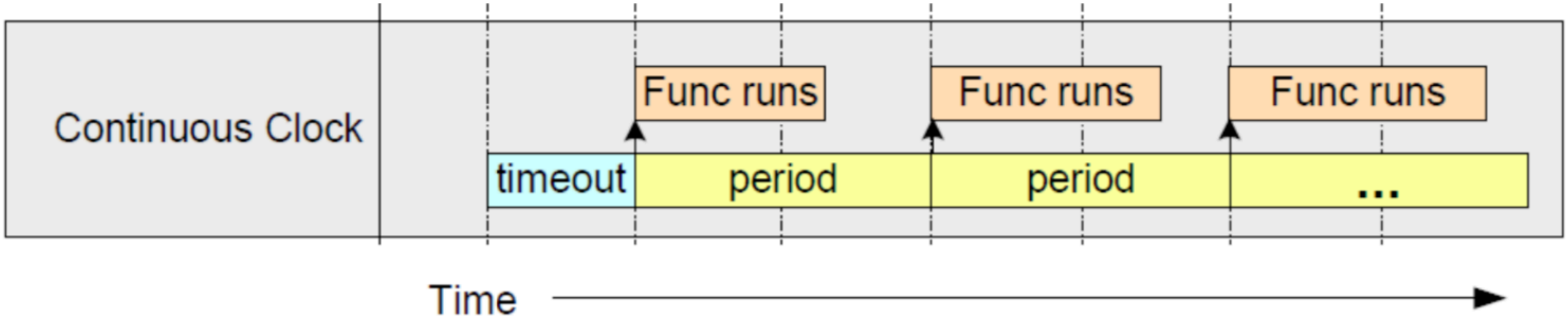
Sets software interrupt of the module to highest priority.

Sets Timer to increment the clock tick count on a periodic basis.

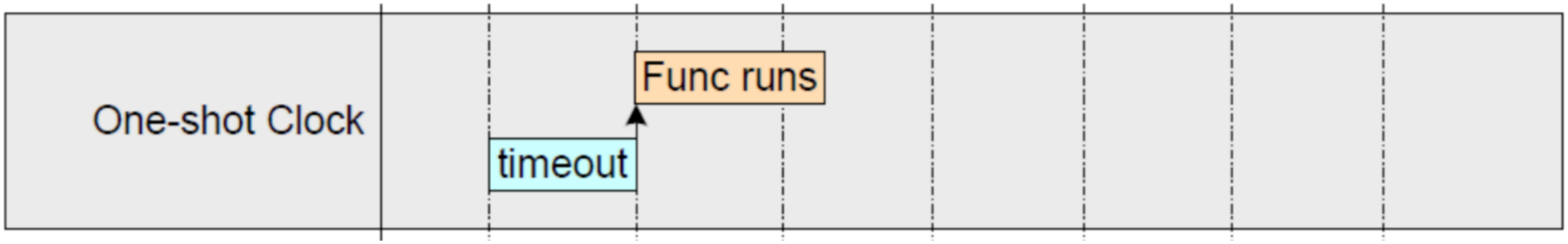
Sets timer period and forces an interrupt on every period.

Clock Module: Calling a Function

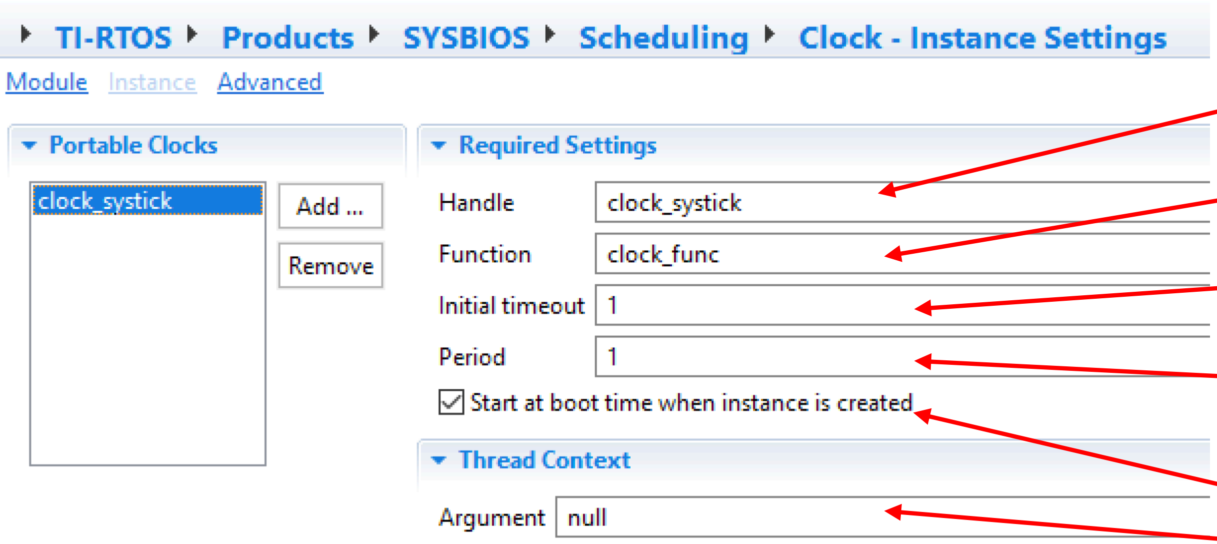
- A clock object can be configured to call functions periodically.
  - This has more accurate timing than sleep as they use a Software Interrupt to call the function and removes the task execution time and task preemption variability from the period.
  - The initial timeout, period and function to be called are configurable.



- Each object can be started and stopped to call a function after a specified delay (one shot) by setting the Period to 0.
  - Functions for starting and stopping a clock object are clock_start() and clock_stop() can also be used to retrigger the one shot.



Clock Object Instance Configuration



▸ TI-RTOS ▸ Products ▸ SYSBIOS ▸ Scheduling ▸ Clock - Instance Settings

▾ Required Settings

Handle: clock_systick

Function: clock_func

Initial timeout: 1

Period: 1

☑ Start at boot time when instance is created

▾ Thread Context

Argument: null

Clock object instance name.

Function to call.

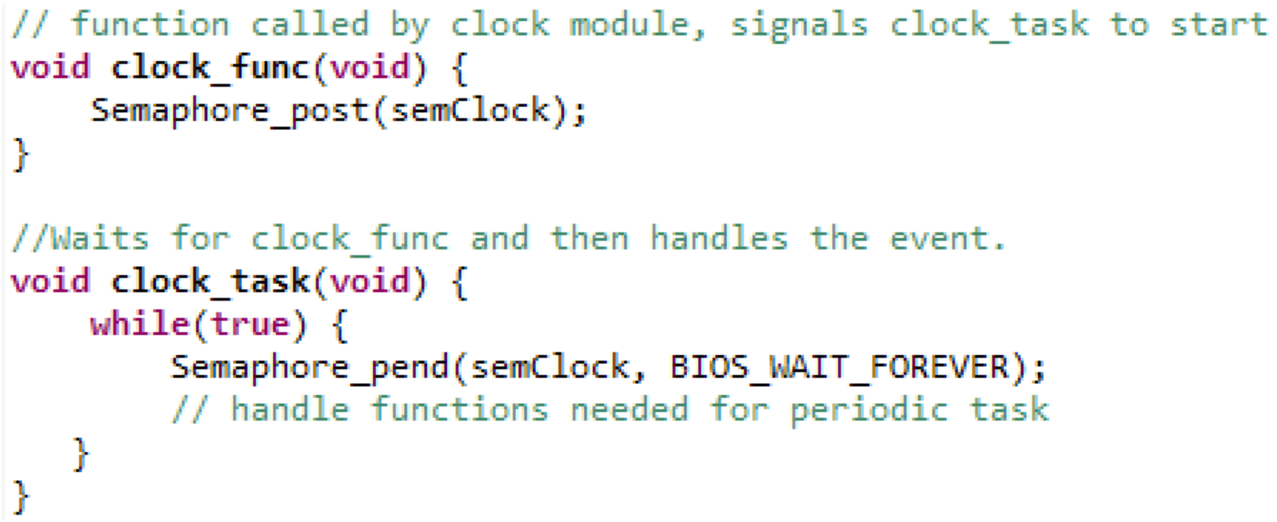
Initial timeout period in clock ticks.

Period of function call in clock ticks.
Period = 0 is one shot mode.

Enables object to start at boot.

Function arguments.

- Example Usage
  - Clock module calls clock_func every 1 msec using a high priority Swi.
  - clock_func() signals clock_task() to unblock the periodic task using a semaphore.



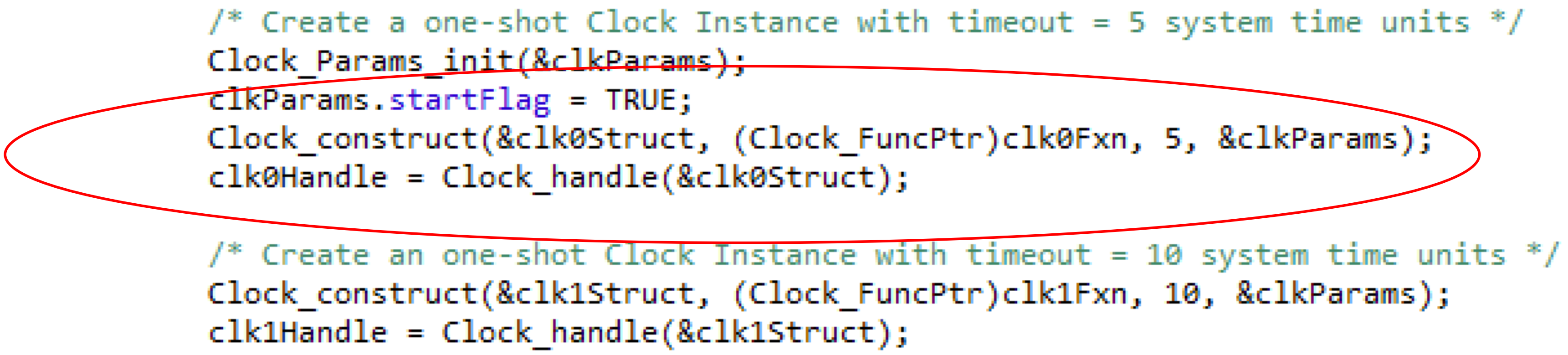
```
// function called by clock module, signals clock_task to start
void clock_func(void) {
    Semaphore_post(semClock);
}

//Waits for clock_func and then handles the event.
void clock_task(void) {
    while(true) {
        Semaphore_pend(semClock, BIOS_WAIT_FOREVER);
        // handle functions needed for periodic task
    }
}
```

Example: ece3849_event

- main.c functionality
  - Shows an example of how to configure objects without the GUI and starts the RTOS.
  - Ece3849_event_cfg shows the RTOS equivalent program.
- It creates an event instance with three events.
  - Event_Id_00 is signaled by an Event_post() in clk0Fxn.
    - clk0Fxn is called from a clock instance with a timeout of 5 system time units and operated in one shot mode.
  - Event_Id_01 is signaled by a Semaphore_post() in clk1Fxn.
    - clk1Fxn is called from a clock instance with a timeout of 10 system time units and operated in one shot mode.
  - Event_Id_02 is signaled by a Mailbox when there is a message ready to read (after a Mailbox_post).
- It starts two tasks.
  - readertask() pends on the event status and prints messages on what events happens.
  - writertask() posts three messages to the Mailbox to trigger Event_Id_02.
- It uses System_printf() to display messages to the console.

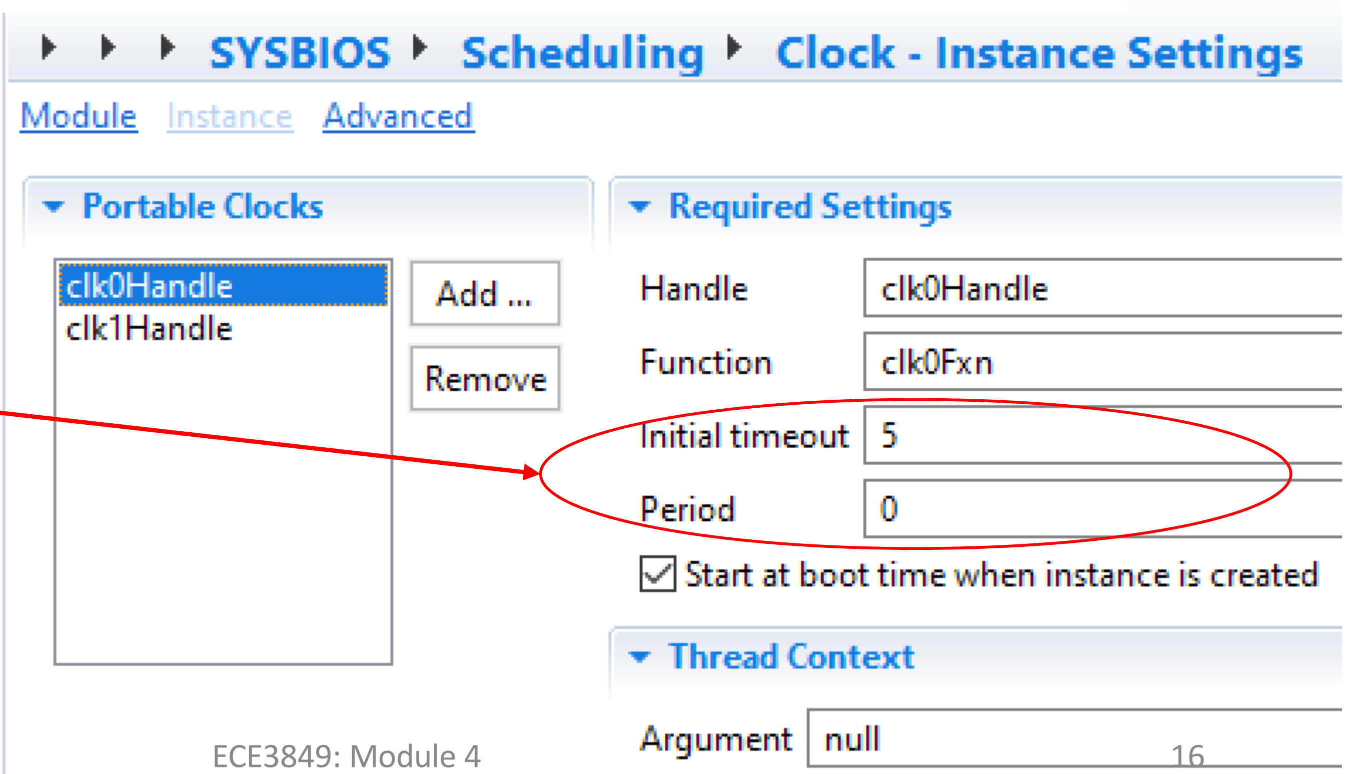
Ece3849_event: Setting up timers

- Configuring a clock instance manually.



```
/* Create a one-shot Clock Instance with timeout = 5 system time units */
Clock_Params_init(&clkParams);
clkParams.startFlag = TRUE;
Clock_construct(&clk0Struct, (Clock_FuncPtr)clk0Fxn, 5, &clkParams);
clk0Handle = Clock_handle(&clk0Struct);

/* Create an one-shot Clock Instance with timeout = 10 system time units */
Clock_construct(&clk1Struct, (Clock_FuncPtr)clk1Fxn, 10, &clkParams);
clk1Handle = Clock_handle(&clk1Struct);
```

- RTOS Equivalent.



clk0Handle
clk1Handle

Handle: clk0Handle

Function: clk0Fxn

Initial timeout: 5

Period: 0

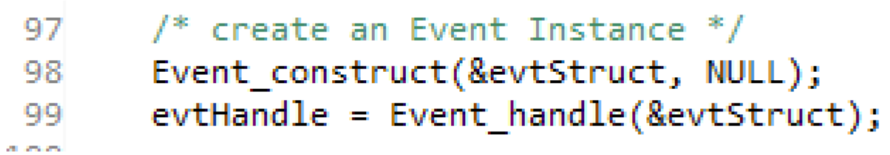
☑ Start at boot time when instance is created

Argument: null

Sets timeout to 5 msec with a Period of 0 for one shot.

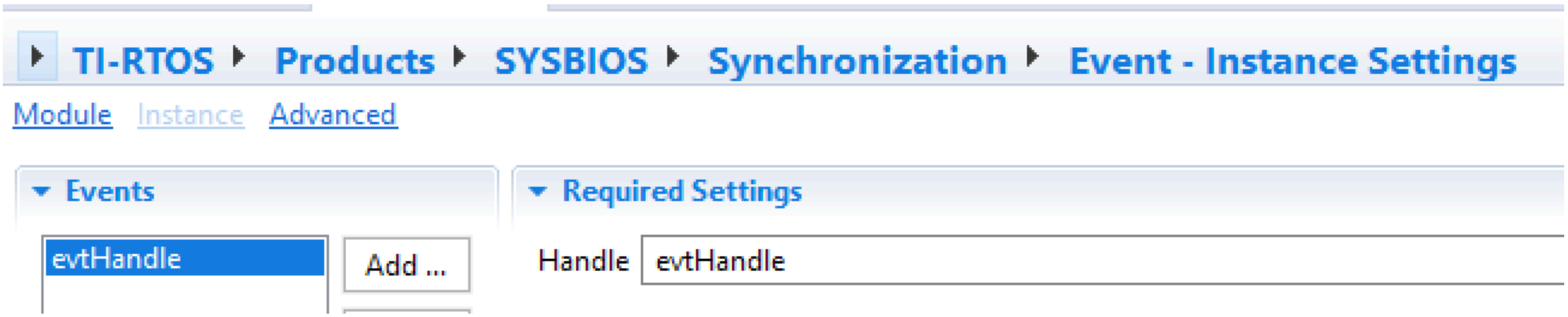
Ece3849_event: Event_Id_00

- Configuring an Event Manually.



```
    /* create an Event Instance */
    Event_construct(&evtStruct, NULL);
    evtHandle = Event_handle(&evtStruct);
```

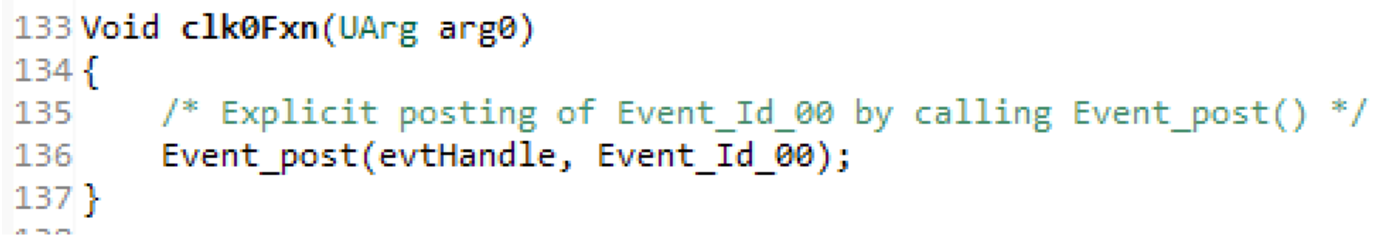
- RTOS Equivalent.


TI-RTOS ▸ Products ▸ SYSBIOS ▸ Synchronization ▸ Event - Instance Settings
Module
Instance
Advanced
Events
Add ...
Required Settings

evtHandle

Handle evtHandle

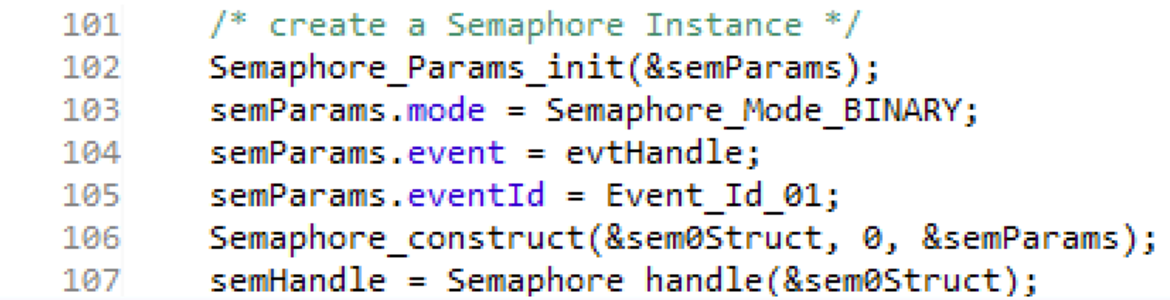
- Signaling an Event_Id_00 with Event_post().



```
Void clk0Fxn(UArg arg0)
{
    /* Explicit posting of Event_Id_00 by calling Event_post() */
    Event_post(evtHandle, Event_Id_00);
}
```

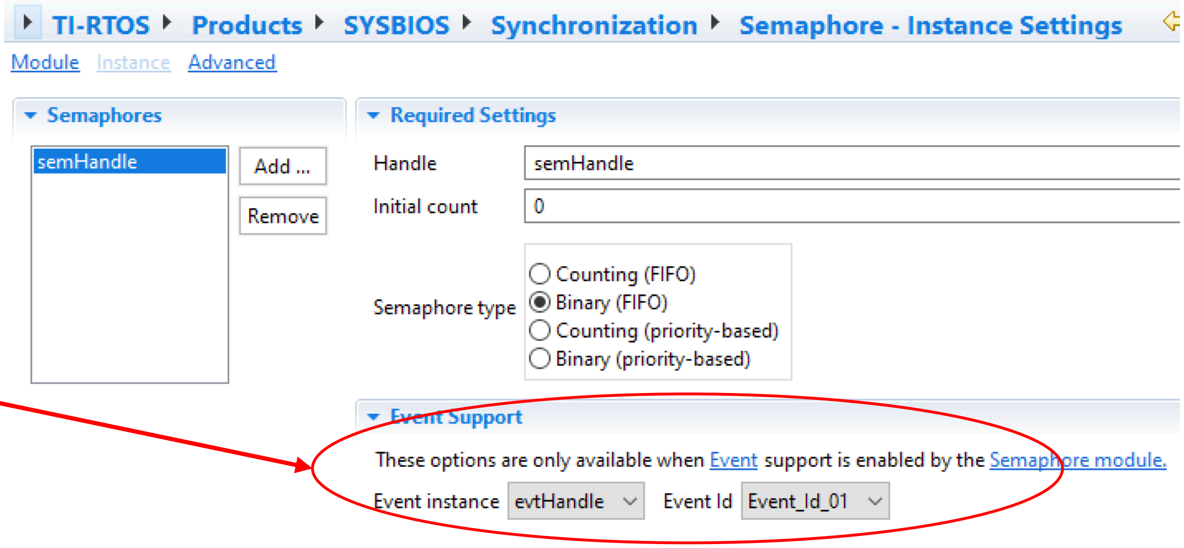
Ece3849_event: Event_Id_01

- Configuring a Semaphore with events manually.



```
    /* create a Semaphore Instance */
    Semaphore_Params_init(&semParams);
    semParams.mode = Semaphore_Mode_BINARY;
    semParams.event = evtHandle;
    semParams.eventId = Event_Id_01;
    Semaphore_construct(&sem0Struct, 0, &semParams);
    semHandle = Semaphore_handle(&sem0Struct);
```

- RTOS Equivalent.


TI-RTOS ▸ Products ▸ SYSBIOS ▸ Synchronization ▸ Semaphore - Instance Settings
Module
Instance
Advanced
Semaphores
Add ...
Remove
Required Settings
Event Support

semHandle

Handle: semHandle

Initial count: 0

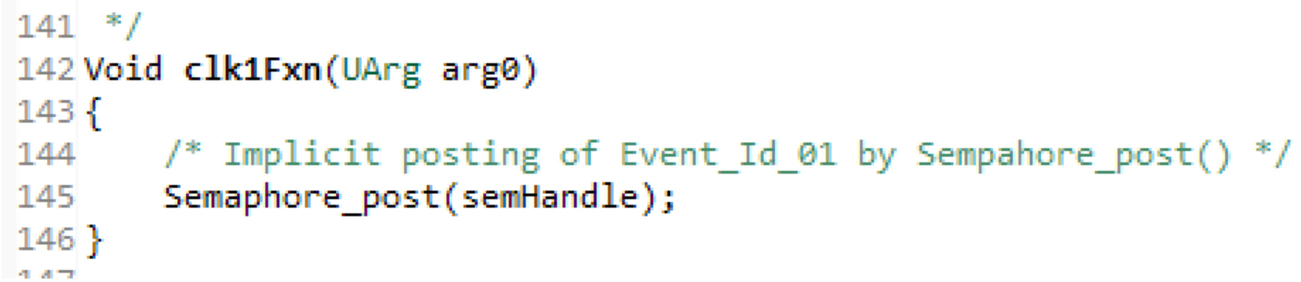
Semaphore type:
○ Counting (FIFO)
● Binary (FIFO)
○ Counting (priority-based)
○ Binary (priority-based)

These options are only available when Event support is enabled by the Semaphore module.

Event instance: evtHandle   Event Id: Event_Id_01

Sets event Handle and Event ID.

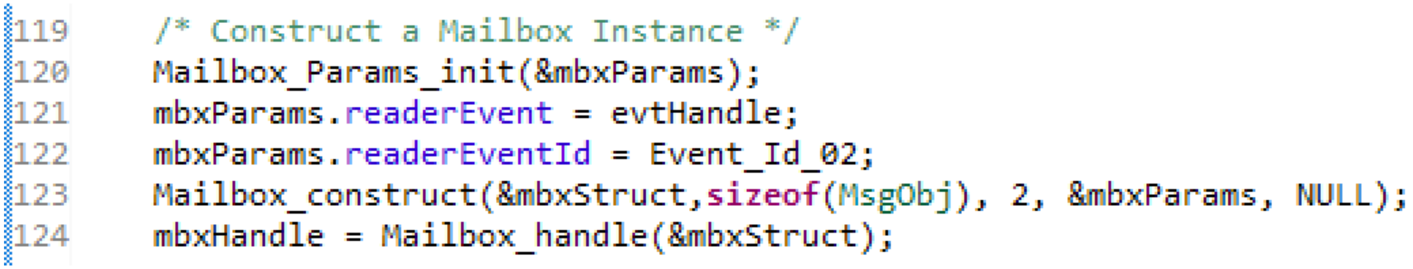
- Signaling an Event_Id_01 with Semaphore_post().



```
 */
Void clk1Fxn(UArg arg0)
{
    /* Implicit posting of Event_Id_01 by Sempahore_post() */
    Semaphore_post(semHandle);
}
```

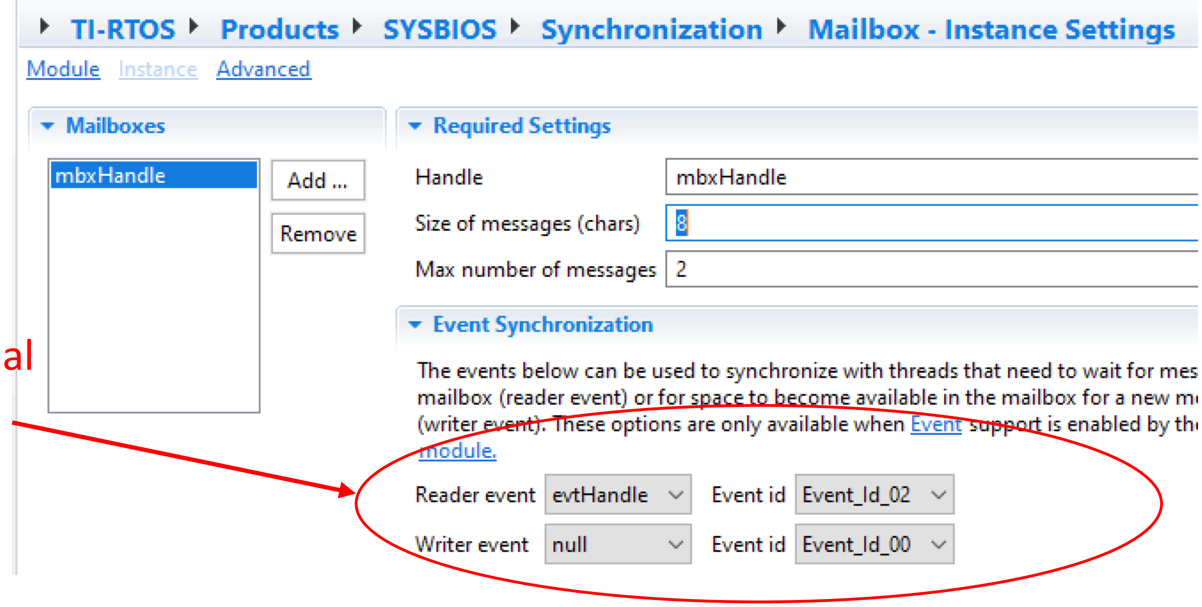
Event3849_event: Event_Id_02

- Configuring a Mailbox with events manually.



```
    /* Construct a Mailbox Instance */
    Mailbox_Params_init(&mbxParams);
    mbxParams.readerEvent = evtHandle;
    mbxParams.readerEventId = Event_Id_02;
    Mailbox_construct(&mbxStruct,sizeof(MsgObj), 2, &mbxParams, NULL);
    mbxHandle = Mailbox_handle(&mbxStruct);
```

- RTOS Equivalent.



mbxHandle

Handle: mbxHandle

Size of messages (chars): 8

Max number of messages: 2

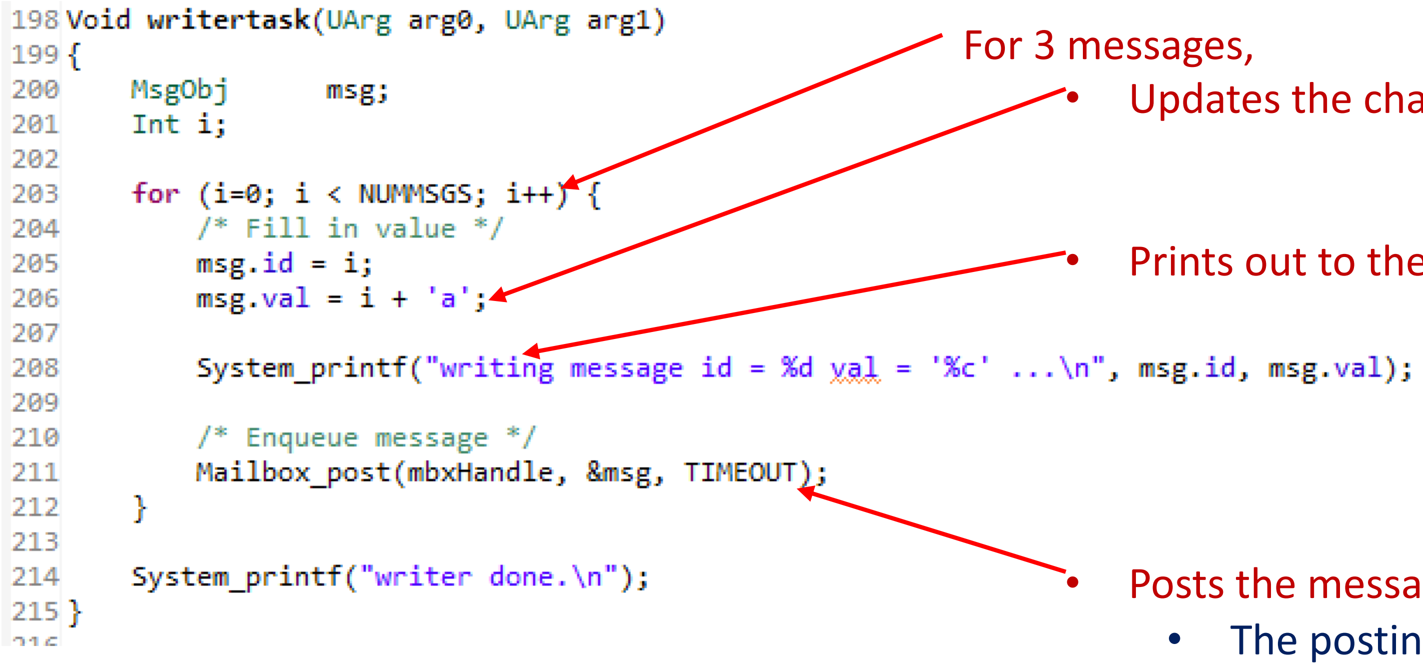
The events below can be used to synchronize with threads that need to wait for mes mailbox (reader event) or for space to become available in the mailbox for a new m (writer event). These options are only available when Event support is enabled by th module.

Reader event: evtHandle — Event id: Event_Id_02

Writer event: null — Event id: Event_Id_00

Sets event Handle, Event ID to signal when there is something to read .

Event3849_event: Mailbox Writer

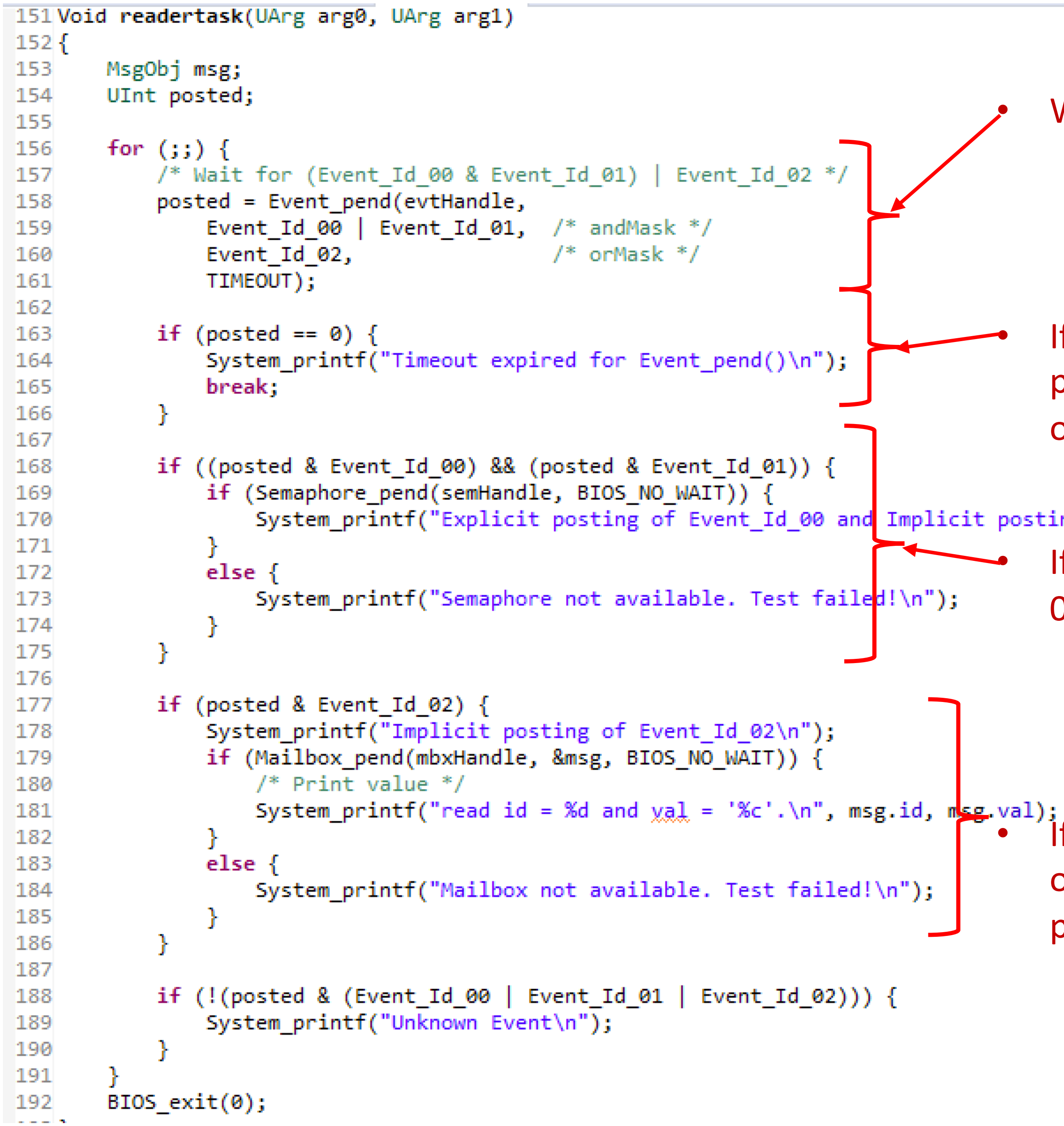


```
Void writertask(UArg arg0, UArg arg1)
{
    MsgObj      msg;
    Int i;

    for (i=0; i < NUMMSGS; i++) {
        /* Fill in value */
        msg.id = i;
        msg.val = i + 'a';

        System_printf("writing message id = %d val = '%c' ...\n", msg.id, msg.val);

        /* Enqueue message */
        Mailbox_post(mbxHandle, &msg, TIMEOUT);
    }

    System_printf("writer done.\n");
}
```

For 3 messages,

- Updates the character to send.
- Prints out to the console.
- Posts the message to the Mailbox.
  - The posting action triggers a Reader event.

Event3849_event: Reader Task

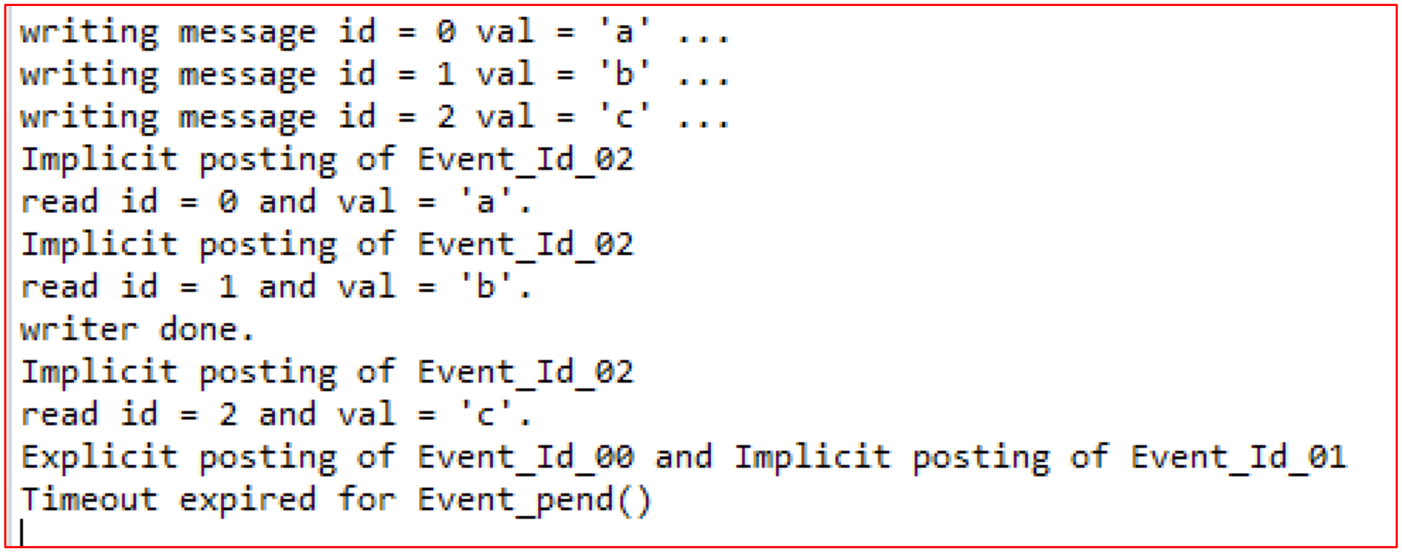


```
Void readertask(UArg arg0, UArg arg1)
{
    MsgObj msg;
    UInt posted;

    for (;;) {
        /* Wait for (Event_Id_00 & Event_Id_01) | Event_Id_02 */
        posted = Event_pend(evtHandle,
            Event_Id_00 | Event_Id_01,  /* andMask */
            Event_Id_02,                /* orMask */
            TIMEOUT);

        if (posted == 0) {
            System_printf("Timeout expired for Event_pend()\n");
            break;
        }

        if ((posted & Event_Id_00) && (posted & Event_Id_01)) {
            if (Semaphore_pend(semHandle, BIOS_NO_WAIT)) {
                System_printf("Explicit posting of Event_Id_00 and Implicit posting of Event_Id_01\n");
            }
            else {
                System_printf("Semaphore not available. Test failed!\n");
            }
        }

        if (posted & Event_Id_02) {
            System_printf("Implicit posting of Event_Id_02\n");
            if (Mailbox_pend(mbxHandle, &msg, BIOS_NO_WAIT)) {
                /* Print value */
                System_printf("read id = %d and val = '%c'.\n", msg.id, msg.val);
            }
            else {
                System_printf("Mailbox not available. Test failed!\n");
            }
        }

        if (!(posted & (Event_Id_00 | Event_Id_01 | Event_Id_02))) {
            System_printf("Unknown Event\n");
        }
    }
    BIOS_exit(0);
```

- Waits for
    - (Event_Id 00 && Event_Id_01)
    - OR Event_Id_02
    - OR TIMEOUT (12 msec)
- If none of the conditions are met, posted equals 0x0, a timeout occurred.
- If posted returns 0011 => 0x3, Event 0 and event 1 occurred.
    - Pends on the Semaphore with without a wait to verify the status and clear the semaphore.
- If post returns 0100 => 0x4, Event 2 occurred, then it gets message and prints the value.

Ece3849_event: Expected Results

- Program Sequence
  - The writertask writes three characters into the Mailbox and writes a message.
  - The readertask processes each of the characters and prints a messages.
  - It then waits for Event_Id_00 and 01 are signaled and prints a message.
  - No other events occur and the event times out.



```
writing message id = 0 val = 'a' ...
writing message id = 1 val = 'b' ...
writing message id = 2 val = 'c' ...
Implicit posting of Event_Id_02
read id = 0 and val = 'a'.
Implicit posting of Event_Id_02
read id = 1 and val = 'b'.
writer done.
Implicit posting of Event_Id_02
read id = 2 and val = 'c'.
Explicit posting of Event_Id_00 and Implicit posting of Event_Id_01
Timeout expired for Event_pend()
```