ECE3849
D-Term 2021

Real Time Embedded Systems

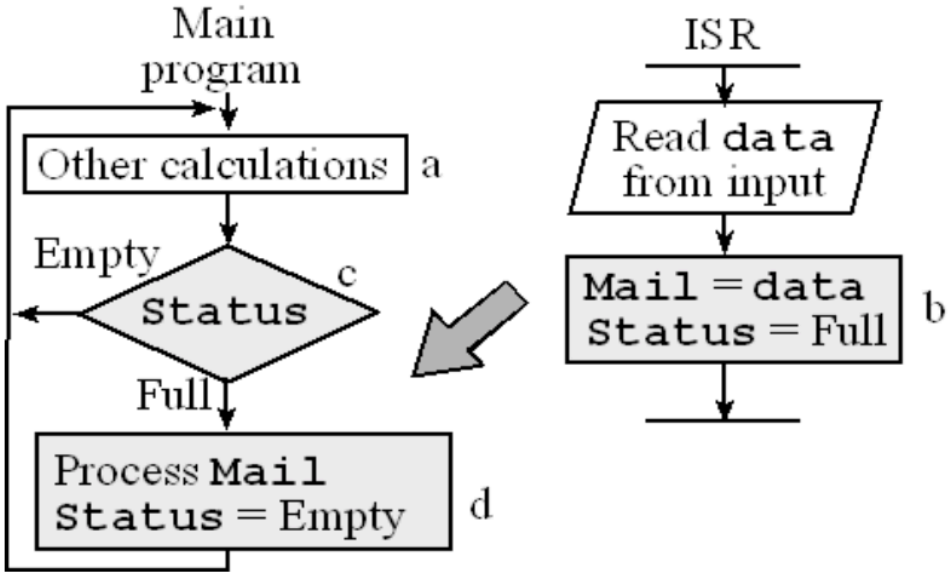
Module 4 Part 1

Module 4 Part 1 Overview

- Inter-task Communication Objects.
  - Message mailboxes.
  - Message queues.
  - Pipes.

Mailbox Review

- Mailboxes are binary semaphores, that protect a shared data variable referred to as a message.
  - They have a Status flag, indicating if the mailbox is empty or full.
  - When the message is written to the mailbox the flag is set to full.
  - When message is read from the mailbox the flag is set to empty.



Mailboxes and Queues

- Queues can contain multiple messages, while mailboxes contain just one message.
  - Both are configured in the TI-RTOS Mailbox Object.
- Both allow exchange of fixed size messages without shared data problems.
  - The size of the messages are configured in the TI-RTOS.
  - Small messages are copied when sending and receiving.
  - For large messages, copying data can be time consuming.
    - Pointers can be cast to int and data can be passed by reference into the mailbox.
    - This may result in shared data issues if the data pointed to changes while it is waiting in the mailbox.
- Mailboxes are implemented using Semaphore objects and therefore have many of the same characteristics.
  - Multiple tasks can read and write into the same queue / mailbox just like with semaphores.
  - Depending on the RTOS implementation tasks can be unblocked in FIFO or priority order.

TI-RTOS Mailbox functions

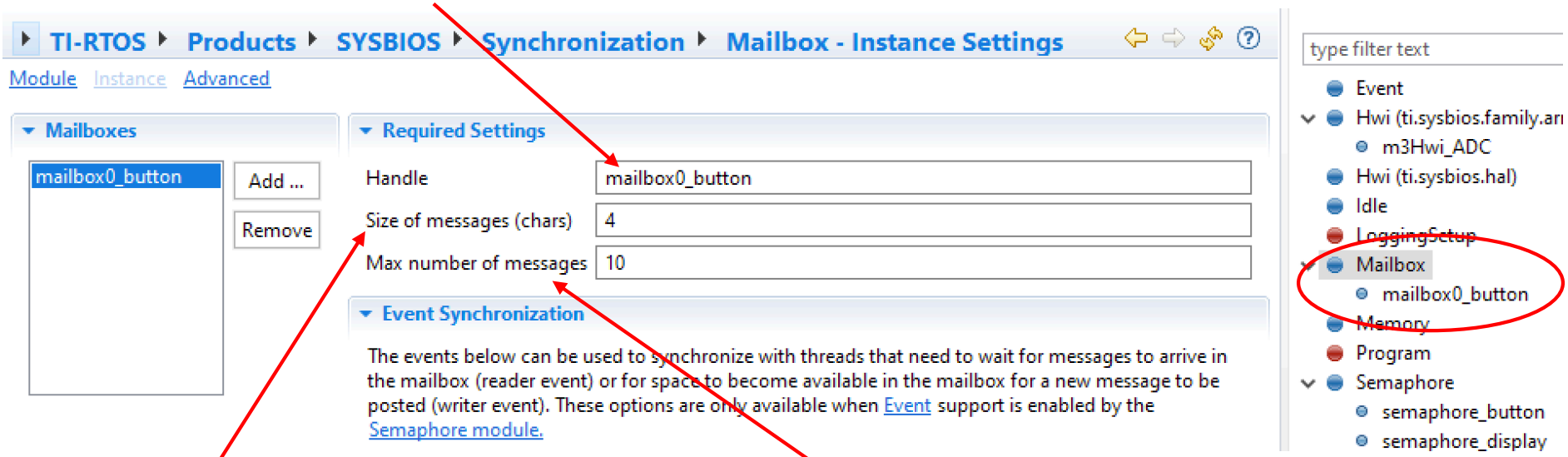
- TI-RTOS implements Mailbox functionality uses two commands.
  - Mailbox_post(): Sends a messages, writing data into the mailbox.
  - Mailbox_pend(): Receives a message, reading data out of the mailbox.
- Mailbox_post() and Mailbox_pend() commands can be configured to be blocking.
  - BIOS_WAIT_FOREVER argument is used when being called from tasks.
  - A pend on an empty queue/mailbox will block.
    - A post to an empty queue/mailbox unblocks a waiting task.
  - A post on a full queue / mailbox will block.
    - A pend on a full queue/mailbox unblocks a waiting task.
- Interrupts need to use non-blocking arguments to the pend and post commands.
  - BIOS_NO_WAIT argument is used in TI-RTOS to accomplish this.
- Syntax Example

```
Mailbox_post(mailbox0, &msg, BIOS_WAIT_FOREVER);
Mailbox_pend(mailbox0, &msg, BIOS_WAIT_FOREVER);
```

Mailbox / Queue Configuration

- Mailboxes and queues are configured with the Mailbox Object in the TI-RTOS.

Name of mailbox



Size of message in number of characters.

Number of messages in the mailbox / queue.

- 1 it is called a Mailbox.
- > 1 it is called a Queue.

TI-RTOS Documentation Mailbox_post() Mailbox_pend()

```
Bool Mailbox_post(Mailbox_Handle handle, Ptr msg, UInt timeout);
```

Post a message to mailbox

ARGUMENTS

handle — handle of a previously-created Mailbox instance object
msg — message pointer
timeout — maximum duration in system clock ticks

RETURNS

TRUE if successful, FALSE if timeout

DETAILS

Mailbox_post checks to see if there are any free message slots before copying msg into the mailbox. Mailbox_post readies the first task (if any) waiting on the mailbox. If the mailbox is full and a timeout is specified the task remains suspended until Mailbox_pend is called or the timeout expires.

A timeout value of BIOS_WAIT_FOREVER causes the task to wait indefinitely for a free slot.

A timeout value of BIOS_NO_WAIT causes Mailbox_post to return immediately.

The timeout value of BIOS_NO_WAIT should be passed to Mailbox_post() to post a message after Event_pend() is called outside of Mailbox_post to wait on an available message buffer.

Mailbox_post's return value indicates whether the msg was copied or not.

```
Bool Mailbox_pend(Mailbox_Handle handle, Ptr msg, UInt timeout);
```

Wait for a message from mailbox

ARGUMENTS

handle — handle of a previously-created Mailbox instance object
msg — message pointer
timeout — maximum duration in system clock ticks

RETURNS

TRUE if successful, FALSE if timeout

DETAILS

If the mailbox is not empty, Mailbox_pend copies the first message into msg and returns TRUE. Otherwise, Mailbox_pend suspends the execution of the current task until Mailbox_post is called or the timeout expires.

A timeout value of BIOS_WAIT_FOREVER causes the task to wait indefinitely for a message.

A timeout value of BIOS_NO_WAIT causes Mailbox_pend to return immediately.

The timeout value of BIOS_NO_WAIT should be passed to Mailbox_pend() to retrieve a message after Event_pend() is called outside of Mailbox_pend to wait on an incoming message.

Mailbox_pend's return value indicates whether the mailbox was signaled successfully.

Pipes

- Similar to queues but do not require a fixed message size.
  - Some are byte-oriented, which send and receive an arbitrary number of bytes.
  - Some allow variable-size messages, but preserve message size from send to receive.
  - Send and receive functions keep track of how many bytes are in each message.
  - Pipes can use memory more efficiently.
    - They can only use the memory needed for that message specific message.
    - Queues need to allocate memory for the the maximum fixed size message.
- Pipes are not available in TI-RTOS

Example FIFO Queue Implementation

- Semaphores used are counting and messages are int.

Initialize semaphores:

Name	Count
RoomLeft	FIFOSIZE
CurrentSize	0
mutex	1

Semaphore used to determine how many locations can be written to.

Semaphore used to determine how many locations can be read from.

Semaphore used to protect critical data section

```
int fifo[FIFOSIZE];   // circular FIFO data structure
int head = 0;
int tail = 0;
```

- Initialize FIFO variables.

```
void Queue_post(int message) {
  Semaphore_pend(RoomLeft, BIOS_WAIT_FOREVER);
  Semaphore_pend(mutex, BIOS_WAIT_FOREVER);
  fifo[tail] = message;  // put onto FIFO
  tail++;                // advance tail index
  if (tail >= FIFOSIZE) tail = 0; // wrap
  Semaphore_post(mutex);
  Semaphore_post(CurrentSize);
}
```

- Check to see if RoomLeft count = 0; Pend if FIFO full.
- Wait for shared resource to become available.
- Write data and calculate your new tail value.
- Release shared resource.
- Increment CurrentSize count to increase number of messages that can be read.

```
void Queue_pend(int *pMessage) {
  Semaphore_pend(CurrentSize, BIOS_WAIT_FOREVER);
  Semaphore_pend(mutex, BIOS_WAIT_FOREVER);
  *pMessage = fifo[head]; // retrieve from FIFO
  head++;                 // advance head index
  if (head >= FIFOSIZE) head = 0; // wrap
  Semaphore_post(mutex);
  Semaphore_post(RoomLeft);
}
```

- Check to see if CurrentSize count =0; Pend if FIFO is empty.
- Wait for shared resource to become available.
- Read data and calculate your new head value.
- Release shared resource.
- Increment RoomLeft count to increase number of messages that can be written.