

FreeRTOS Queues

FreeRTOS allows communication and synchronization between tasks using queues. This is similar to message passing with some limitations.

The queue is not owned by a particular task. Any number of tasks can write to or read from any queue. It normally acts as a FIFO queue.

The queue will hold a finite number of fixed size data items. **Length and item size** are specified during creation of the queue.

Slide 1

FreeRTOS allows information exchange and synchronization among tasks using queues. These are similar to mailboxes, but with limited options as a result of small system size.

Slide 2

Once queue is created, any task can send data to, or receive data from this queue. With few exceptions, queue operates in first-in first-out mode without message priorities.

FreeRTOS Queues



The function `xQueueCreate()` creates a queue and returns a handle to this queue (or NULL if creation fails).

```
#include "queue.h"
.....
QueueHandle_t qh
.....
.....
qh=xQueueCreate(10, 1);
```

Length of queue Size of item in bytes

Slide 4

We need to include `queue.h` to use the following functions. A handle to a queue is a word used to refer to the queue later. The function `xQueueCreate()` takes two arguments: first is the number of items the queue can hold, and the second is the size of each item. The type `QueueHandle_t` is actually a word, but given a specific name to improve program readability.

FreeRTOS Queues

The function `xQueueSend()` is used to **write an item at the back of the queue**. If queue is full, the calling task **will be blocked** until a place in queue is empty. Calling task may **time out** and stop waiting for an empty place.

```
xQueueSend (qh1, &x, 1000);
```

Handle of the queue

Pointer to variable with the value to be written.

Wait for 1000 ticks then time out.

Function returns either `pdPASS (=1)` or `errQUEUE_FULL (=0)`.

FreeRTOS Queues

The function `xQueueReceive()` is used to **read an item** from the front of the queue (and **remove it** from queue). If queue is empty, the calling task will be **blocked** until an item is sent to queue. **Timing out** is also possible.

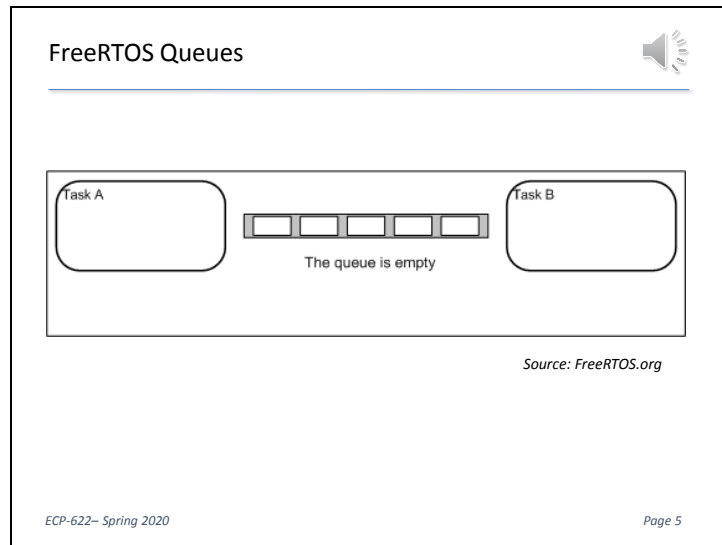
```
xQueueReceive (qh1, &x, 1000);
```

Handle of the queue

Pointer to variable in which value is buffered.

Wait for 1000 ticks then time out.

Function returns either `pdPASS (=1)` or `errQUEUE_EMPTY (=0)`.



Slide 7

This animation shows Task A sending three items to the FIFO queue using `xQueueSendToBack`. `xQueueSendToBack` acts exactly as `xQueueSend`. Then, Task B receives the three items by executing `xQueueReceive` three times.

FreeRTOS Queues

For indefinite waiting in send or receive function, timeout is set to `portMAX_DELAY`.

If multiple tasks are waiting for the same queue, the one unblocked first will be the task with higher priority or task that waited longer.

Other functions are available for queue handling. For example, `xQueueSendToFront()` writes an item at the front of the queue, `xQueuePeek()` reads the value at the front of the queue without removing it, and `uxQueueMessagesWaiting()` returns the number of items currently in the queue.

ECP-622- Spring 2020

Page 6

Slide 8

This option allows waiting with no timing-out, but this should be avoided in real-time tasks.

Slide 10

`xQueueSendToFront` is the only way to give priority to a particular message. `xQueueReceive` removes item from queue after reading but `xQueuePeek` leave it there. Number of items currently in queue can be checked at any time by the indicated function.