EE458 - Embedded Systems
Introduction to uC/OS Cont.

• Outline
  – uC/OS Queues
  – uC/OS FIFOs
  – uC/OS Flags

• References
  – uC/OS RTOS Library Reference
A uC/OS queue is a fixed-size FIFO. Only 32-bit quantities may be stored in the queue. In addition to allocating a uC/OS queue data structure, you must also allocate the memory that is used to hold the messages:

```c
#define QSIZE 10
OS_Q qds;      // Queue data structure
void *QStorage[QSIZE]; // Queue storage
```
UC/OS Queues

Before using the queue you must initialize it:

```c
status = OSQInit(&qds, &QStorage[0], QSIZE);
```

Use `OSQPost()` to post (32 bit) messages to the queue. This function will not block. It will return an error if the queue is full:

```c
status = OSQPost(&qds, (void *) msg);
```
Introduction to uC/OS Cont.  

uC/OS Queues

- **OSQPend()** is used to retrieve the first value from the queue. This function will block if the queue is empty:

  ```c
  BYTE error;
  char *msg = (char *)OSQPend(&qds,timeout,&error);
  ```

- **error** is a return argument and should be checked to see if an error occurred. If **timeout** is 0, the function will wait **forever**.
A FIFO can be used to pass data structures from one task to another. The data structure must have a `(void *)` pointer as its first element:

```c
struct {
   void *ptr;
   char str[20];
} MyStruct;
```

```c
OS_FIFO myfifo;
status = OSFifoInit((OS_FIFO *) &MyStruct);
```
**OSFifoPost()** is used to post data structures to the FIFO:

```c
MyStruct mydata;
strcpy(mydata.str, "Hello world");
status = OSFifoPost
    (&myfifo, (OS_FIFO_EL *) &mydata);
```

**OSFifoPostFirst()** can be used to post a structure at the front of the FIFO:

```c
status = OSFifoPostFirst
    (&myfifo, (OS_FIFO_EL *) &mydata);
```
Introduction to uC/OS Cont.

uC/OS FIFOs

- **OSFifoPend()** is used to retrieve a data structure from a FIFO. It returns a pointer to the posted data structure:

  ```c
  MyStruct *str_ptr;
  str_ptr = (MyStruct *) OSFifoPend(&myfifo, timeout);
  ```

- This function will block until there is data in the FIFO or the **timeout** value is reached. If **timeout** is 0 the function will block forever.
\textbf{Introduction to uC/OS Cont.}

\textbf{uC/OS Queues vs FIFOs}

- uC/OS Queues and FIFOs have some similarities, but also some fundamental differences. Queues should definitely be used when passing 32-bit data items (samples from an A/D converter).

- Either can be used to “pass” larger data structures. (Pass pointers to the data structures when using a queue.) Queues are of fixed size while FIFOs are of arbitrary size (a linked-list underlies a FIFO).
Introduction to uC/OS Cont.

uC/OS Queues vs FIFOs

- Normally messages are added to the end of a queue or a FIFO, but it is possible to add a message to the front of a FIFO.
- Note that the message is not copied when working with a FIFO. You must ensure that access to the underlying data by the sending and receiving tasks is synchronized. (This same problem exists when using a queue to pass pointers to data structures.)
Introduction to uC/OS Cont.

uC/OS Flags

- uC/OS flags are similar to RTEMS events. A flags object must be initialized before using any of the related functions:

  ```c
  OS_FLAGS myflag;
  OSFlagCreate(&myflag);
  ```

- Bits are set using `OSFlagSet()`:

  ```c
  // Set bits 7 and 31
  OSFlagSet(&myflag, 0x80000080);
  ```
You can either wait on all bits in a given mask to be set or on any of them:

// Wait until all of bits 0-3 and 12-15 are set
status = OSFlagPendAll
        (&myflag, 0x0000F00F, timeout);

// Wait until any odd bit is set
status = OSFlagPendAny
        (&myflag, 0xA0000000, timeout);
Introduction to uC/OS Cont.

uC/OS Flags

- **OSFlagClear()** is used to clear flags in a flag object:

  ```c
  // Clear all flags
  OSFlagClear(&myflag, 0xFFFFFFFF);
  ```

- Finally, **OSFlagState()** returns the current flag register:

  ```c
  DWORD cur_flg = OSFlagState(&myflag);
  ```