Microcontroller Based Design

Today's Lecture Plan

- Description of second project
- Term exam comment
- Microcontroller development
- Real-time operating systems
- Application to embedded biomedical systems

Embedded C/C++

- Standard C/C++ language constructs are used to develop applications
- Use a special header file for each device to define its SFRs in plain English
  - P2 instead of 0x1F
- Use a C compiler to generate HEX code ready to download to device

Function vs. Macro

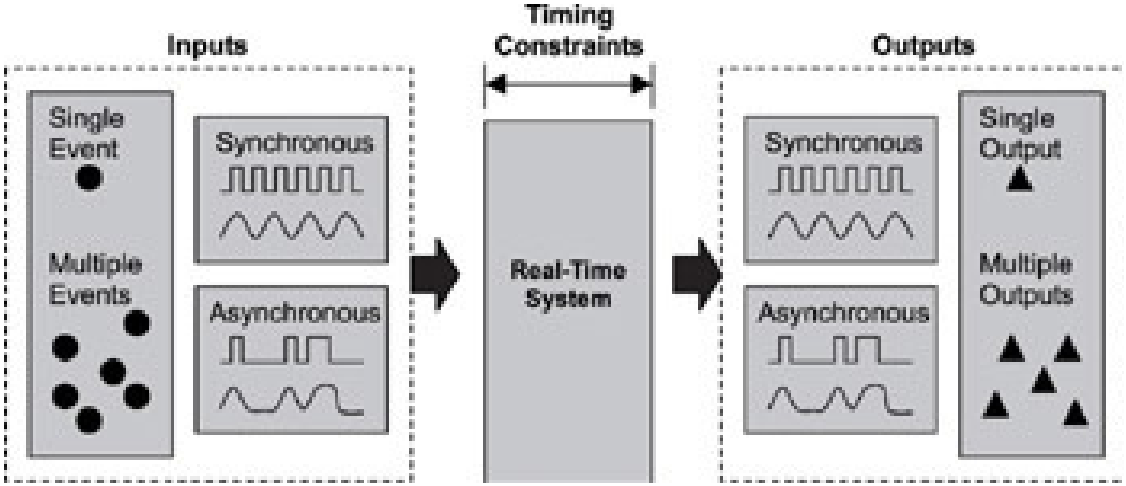
- Functions are subrourines
  - Stored in code once but executed many times
  - Require suitable stack length
  - Slower
- Macros
  - Actual hard-coded segments
  - Stored whenever called in program

Embedded System Definition

- Embedded systems are computing systems with tightly coupled hardware and software integration, that are designed to perform a dedicated function.
- The word embedded reflects the fact that these systems are usually an integral part of a larger system, known as the embedding system.
- Multiple embedded systems can coexist in an embedding system.

Real-Time Embedded Systems

- In the simplest form, real-time systems can be defined as those systems that respond to external events in a timely fashion.
- The response time is guaranteed
- Timing correctness is just as important as functional or logical correctness.



Hard/Soft Real-Time Systems

- A ***hard real-time system*** is a real-time system that must meet its deadlines with a near-zero degree of flexibility.
  - The deadlines must be met, or catastrophes occur.
- A ***soft real-time system*** is a real-time system that must meet its deadlines but with a degree of flexibility.
  - The deadlines can contain varying levels of tolerance, average timing deadlines, and even statistical distribution of response times with different degrees of acceptability.
  - A missed deadline does not result in system failure, but costs rise in proportion to the delay

Real-Time Operating System

- Allows you to create applications that simultaneously perform multiple functions or **tasks**.
- While it is certainly possible to create real-time programs without an RTOS (by executing one or more functions or tasks in a loop) there are numerous scheduling, maintenance, and timing issues that an RTOS can solve for you.
- RTOS allows flexible **scheduling** of system resources like the CPU and memory and offers some way to communicate between tasks.

Basic RTOS Functionality

- The basic functionality allows to start and stop **concurrent tasks** (processes).
- Additional functions support an inter-process communication.
- This communication may be used to synchronize different tasks, to manage common resources like peripherals or memory regions and to pass complete messages between tasks.

Basic Functions

- The basic functions are used to start up the Real-Time Executive, to start and stop tasks and to pass control from one task to another (round-robin scheduling). It is possible to assign execution priorities to tasks. These are used to select one particular task to be run next, if more than one task is ready to run (preemptive scheduling).

Inter-process Communication

- RTOS provides several ways for interprocess communication including:
  - Event flags
  - Semaphores
  - Mutexes
  - Mailboxes.

Event Flags

- The primary means to implement a task synchronization
- Each task has 16 event flags assigned to it and may therefore wait selectively for 16 different events.
- It is possible for a task to wait for more than one flag at the same time. In this case it can be chosen if all selected flags have to be set before the task continues (AND-connection), or the task continues if just one or a few of all selected flags are set (OR-connection).
- Event flags may be set by **interrupt functions** as well.
  - possible to synchronize asynchronous external events to RTOS tasks.

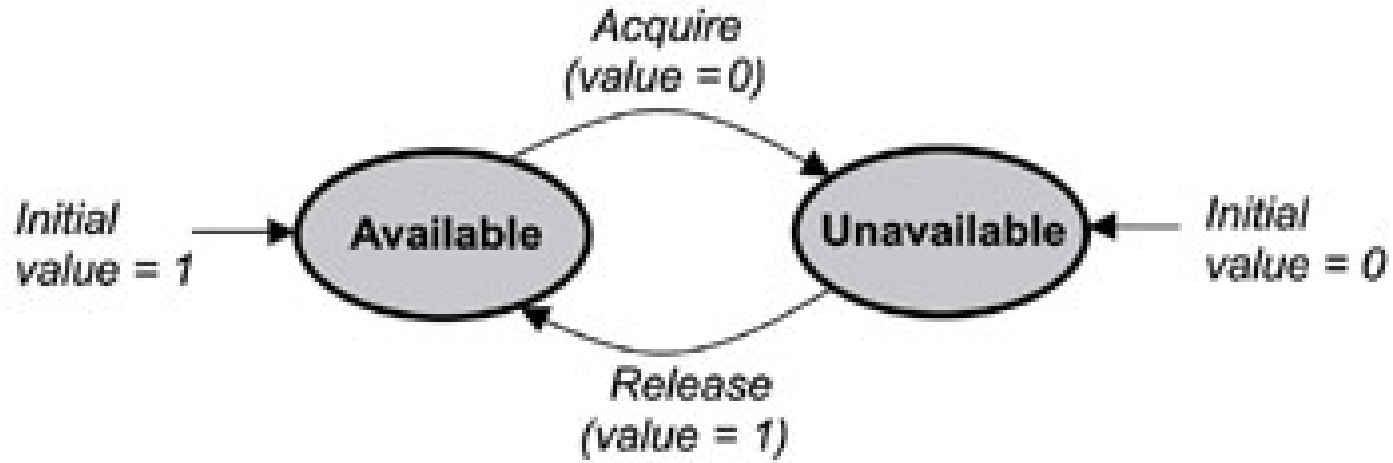
Resource Scheduling

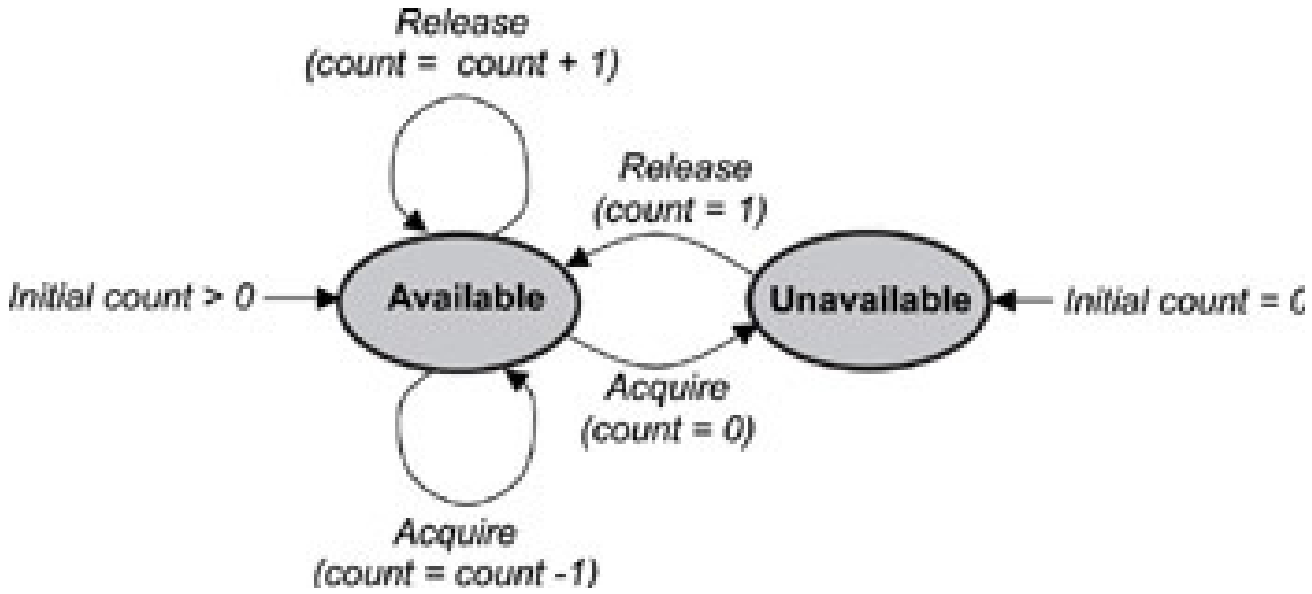
- If a common resource has to be accessed by more than one task, special means are required in a real-time multitasking system.
- Otherwise different accesses may interfere and lead to inconsistent data or a miss-behavioral of a peripheral element.

Semaphores

- The primary means for resource reservation
- These are software objects containing a **virtual token** (binary semaphore).
- The token is passed to one task at a time thus excluding interfering accesses to a common resource.
- The semaphore may put a task to sleep, if the token is not available.
  - It will be waken up as fast as the token is returned to the semaphore.
  - To handle erroneous situations it is possible to combine the wait for a token with a time-out.

Binary vs. Counting Semaphores

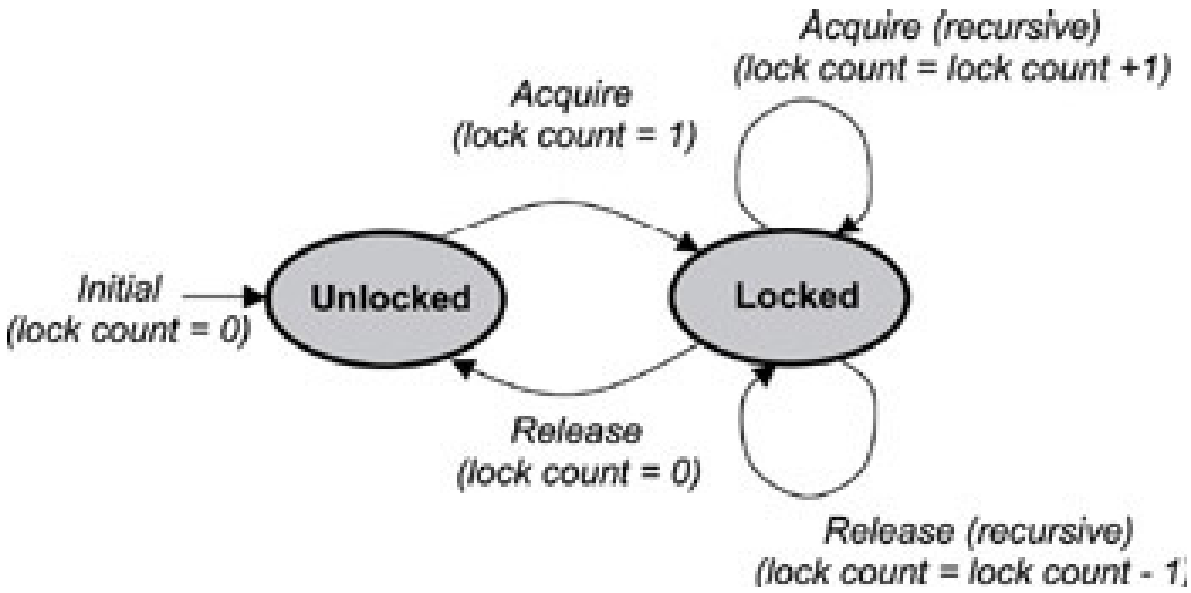




Mutexes

- An alternative approach to synchronization problems is the use of mutual exclusion locks - **mutexes**.
- These are software objects that can be used to lock the common resources and allow access only from a task that owns the mutex.
- The other tasks are blocked until a mutex is released.

Mutex Operation



Mailboxes

- It is sometimes required to exchange **messages** between tasks.
- The message is simply a **pointer** to the block of memory containing a protocol message or frame.
- The memory block is dynamically allocated and provided by the user. It is the user responsibility to properly allocate / deallocate the memory blocks to prevent memory leaks.
- The message may put the task to sleep, if the message for the message waiting task is not available. It will be waken up as soon as the message is sent to the mailbox to the waiting task.

Round-Robin Multitasking

- Round-Robin allows quasi-parallel execution of several tasks.
- Tasks are not really executed concurrently but are **time-sliced** (the available CPU time is divided into time slices and RTOS assigns a time slice to each task).
- Since the time slice is short (only a few milliseconds) it appears as though tasks execute simultaneously.
- Tasks execute for the duration of their time-slice (unless the task's time slice is given up). Then, RTX Kernel switches to the next task that is **ready** to run and has the **same priority**.
  - If no other task with the same priority is ready to run, the currently running task resumes it's execution.

Cooperative Multitasking

- You must call the system wait or pass functions somewhere in each task.
- These functions signal RTOS to switch to another task.

Preemptive Multitasking

- If a task with a higher priority than currently running task becomes ready to run, it will **suspend** the current running task.
- Examples include:
  - An **event** is set for a higher priority task by a currently running task or by an interrupt service routine. The higher priority task continues to run.
  - a token is returned to a **semaphore** and a higher priority task is waiting for one. The semaphore waiting task will continue to run.
  - A **mutex** is released and a higher priority task is waiting for it. The currently running task will be suspended and a task waiting for mutex will continue to run.
  - A message is posted to a **mailbox** and a higher priority task is waiting for one. The currently running task will be suspended and a message waiting task will continue to run.
  - The **priority** of currently running task has reduced. If other task is ready to run and has a higher priority than currently running task, this task is suspended immediately and higher priority task resumes it's execution.

Applications

- Design of a biomedical embedded system
  - Block diagram
  - Development of hardware drivers
  - Generating high-level description of tasks and their timing
  - Use RTOS whenever available to simplify design tasks scheduling
  - Choose the right development tools