EE458 - Embedded Systems Modularization

● Outline
  - Decomposing Applications
  - Final Projects

● References
  - RTC: Chapter 14
Modularization
Decomposing Applications

- How do we break an application into concurrent tasks? How do we assign task priorities?
- There are several approaches, we will focus on the outside-in decomposition method.
- In the outside-in method we first identify the input and output devices and then assign tasks to handle each device following some general guidelines.
Modularization
Decomposing Applications

- Guidelines to outside-in decomposition:
  1) Identify Device Dependencies
     - 1a) Identify Active I/O Devices
     - 1b) Identify Passive I/O Devices
  2) Identify Event Dependencies
  3) Identify Time Dependencies
     - 3a) Identify Critical and Urgent Activities
     - 3b) Identify Different Periodic Execution Rates
     - 3c) Identify Temporal Cohesion
Modularization
Decomposing Applications

- Guidelines to outside-in decomposition (cont):
  - 4) Identify Computationally Bound Activities
  - 5) Identify Functional Cohesion
  - 6) Identify Tasks with a Specific Purpose
  - 7) Identify Sequential Cohesion
Modularization

1) Identify Device Dependencies

- Active I/O devices are those that generate interrupts to communicate with the application. Synchronous devices generate interrupts periodically or in sync with other active devices, asynchronous devices do not.

- Passive I/O devices do not generate interrupts. These devices must be polled. Communication can be periodic or aperiodic.
Modularization

1a) Identify Active Devices

- Recommendations
  - 1) Initially assign separate tasks for separate asynchronous I/O devices.
  - 2) Combine tasks for I/O devices that generate infrequent interrupts having long deadlines.
  - 3) Assign separate tasks to devices that have different input and output rates. Devices with high I/O frequencies should be associated with higher priority tasks.
Modularization

1a) Identify Active Devices

- Recommendations (cont.)
  - 4) Assign higher priorities to tasks associated with interrupt generating devices. The task must be able to execute fast enough to keep up with the interrupts.
  - 5) Assign a resource control task to control access to I/O devices. This task can receive multiple I/O requests from different tasks.
  - 6) Assign an event dispatch task for I/O device requests that need to be handed off to multiple tasks. May handle one or more devices.
Modularization

1b) Identify Passive Devices

• Recommendations
  – 1) Assign a single task to interface with passive devices when communication with such devices is aperiodic and when deadlines are not urgent.
  – 2) Assign separate polling tasks to send periodic requests to passive I/O devices.
  – 3) Trigger polling requests via timer events. Do not use busy loops, they are subject to interrupts and preemption.
  – 4) Assign a high relative priority to polling tasks with relatively short periods.
Modularization

2) Identify Event Dependencies

- Externally generated events are interrupts. Tasks will also need to be assigned to handle internally generated events such as error conditions or faults.

- Should the event be sent to a task for user notification via an I/O device? Can corrective action be taken to clear the fault?
3) Identify Time Dependencies

- 3a) Identify Critical and Urgent Activities
  - Failure of a critical task to meet a deadline is disastrous. Priorities for critical and urgent tasks should be relatively high.

- 3b) Identify Periodic Execution Rates
  - Activities with similar rates can be grouped into tasks.

- 3c) Identify Temporal Cohesion
  - Group sequences of code that always execute at the same time into a single task.
Modularization

4) Ident. Comp. Bound Activities

- Computationally bound activities require a lot of CPU time compared to the time required for other operations, such as I/O.
- These are typically number crunching activities and may have relatively long deadlines. Typically low priorities are assigned to the corresponding tasks so that they do not monopolize the CPU. These types of tasks may be time-sliced at a common (low) priority level.
5) Identify Function Cohesion
   - Group functions that perform related activities into a single task. Two tasks that pass lots of data may be combined into a single task.

6) Identify Tasks with a Specific Purpose
   - Tasks can be grouped according to the purpose they serve. Should there be a safety task to detect problems, set alarms, and notify the user?

7) Identify Sequential Cohesion
   - Group activities that must occur in a given sequence into a single task.
Modularization
Final Projects

- Demonstrate an RTOS (other than one we use in class) running on real or simulated hardware. Examples: FreeRTOS, eCOS.
- Demonstrate RTEMS, uC/OS, AvrX running on hardware different from that used in class.
- Demonstrate an RTEMS, uC/OS, or AvrX application.
Modularization
Final Projects

- Example RTEMS projects:
  - Networking
  - Remote Debugging
  - File Systems
  - Device Driver Development
  - Graphical User Interface
  - Eclipse or CodeBlocks IDE
  - Ports to PowerPC, ARM, Coldfire

- Some of the research projects could develop into Senior projects ...