Embedded Systems Chapter Notes

Computers are used to control a wide range of systems from simple domestic machines, through games controllers, to entire manufacturing plants. These computers interact directly with hardware devices. Their software must react to events generated by the hardware and, oten, issue control signals in response to these events. These signals results in an action, such as the initiation of a phone call, the movement of a charater on the screen, the openining of a valve, or the display of the system status.

SW in these systems is embedded in systems hardware, often in read-only memory, and usually responds, in real time, to events from the system’s environment. By real time, I mean that the software has a deadline for responding to external events. If this deadline is missed, then the overall hardware-software system will not operate correctly.

Embedded SW is very important economically because almost every electrical device now includes software. There are therefore, more embedded software systems that other types of SW system.

* Examples of embedded systems
  + Phones
  + Computers
  + Microwaves

Responsiveness in real time is the critical difference between embedded systems and other software systems such as information systems. For non-real-time systems, the correctness of a system can be define by specifying how system inputs map to corresponding outputs that should be produced by the system. In response to an input, a corresponding output should be generated by the system, and often, some data should be stored. However, in a real-time systems, the correctness depends both on the response to an input and the time taken to generate that response. If the system takes too long to respond, then the required response may be ineffective.

* Example
  + If embedded software controlling a car braking system is too slow, then an accident may occur because it is impossible to stop the car in time

Definition of a Real-Time Software System:

* A real-time SW system is a system whose correct operation depends on both the results produced by the system and the time at which these results are produced. A “soft real-time system” is a system whose operation is degraded if results are not produced according to the specified timing requirements. IF results are not produced according to the timing specification in a “hard real-time system”, this is considered to be a system failure.

Timely response is an important factor in all embedded systems, but not all embedded system require a very fast response.

Apart from real-time response, here are more differences between embedded systems and other types of systems:

* Embedded systems generally run continuously and do not terminate. They begin/end when the hardware is turned on/off. -> Reliable software engineering techniques must be used to ensure continuous operation
* Interactions with the system’s environment are uncontrollable and unpredictable. Unlike, other systems, embedded systems must respond to unexpected events at any time. In other systems, the pace of interaction if controlled by the system, and, by limiting user options, the events to be processed are known in advance. -> This leads to a design for real-time systems based on concurrency, with several process executing in parallel
* There may be physical limitations that affect the design of the system.
  + Examples
    - Power available to the system
    - Physical space taken up by the hardware
* Direct hardware interaction may be necessary. Others systems have a layer of software (device drivers) that hides the hardware from the OS. This is possible because you can only connect a few types of device to these systems like keyboards, mice, displays, etc. By contrast, embedded systems may have to interact with a wide range of hardware devices that do not have separate device drivers.
* Issues of safety and reliability may dominate the system design. Many embedded system control devices whose failure may have high human or economic costs.

Embedded Systems Design (Section 20.1)

* Design process for embedded systems is a systems engineering process in which the SW designers have to consider in detail the design and performance of the system hardware.
  + Process may involve deciding which capabilities are to be implemented by SW or HW.
* In most embedded systems (like phones), the costs and power consumption of hardware are critical to design
  + Sometimes special-purpose hardware may have to be designed and built
* \*\*Top-down approach for the SW design process is impractical for most real-time systems.
  + Low-level decisions on hardware, support SW, and system timing must be considered early in the process
* Most general approach to embedded, real-time SW design is based on a stimulus-response model
  + This is because embedded systems are reactive systems that react to events in their environment
  + Stimulus = event in environment that causes SW to act in some way
  + Response = signal or message sent by SW to the environment
  + You can define the behavior of a real-time system by listing the stimuli received by the systems, the associated responses, and the time at which the response must be produced.
  + 2 types of Stimuli
    - Periodic Stimuli
      * Occur at predictable time intervals
    - Aperiodic Stimuli
      * Occur irregularly and unpredictably and are usually signaled by the computer’s interrupt mechanism
  + General design guideline is to have separate processes for each type of sensor and actuator (responses are sent to actuators).
    - Actuators receive responses from the real-time system.
      * They control equipment (such as a pump) when then makes changes to the system’s environment
      * Actuators can also generate stimuli
    - For each type of sensor, there may be a sensor management process that handles data collection from the sensors.
    - Data processing processes compute the required responses for the stimuli received by the system (See figure 20.3)
    - Actuator control processes are associated with each actuator and manage the operation of that actuator
  + A real-time system has to respond to stimuli that occur at different times -> you must then organize the system so that as soon as a stimulus is received, control is transferred to the correct handler.
    - This is impractical in sequential programs
    - Consequently, real-time SW systems are normally designed as a set of concurrent, cooperating processes
      * These are usually supported by a real-time operating system
  + Typical SW Design Process includes
    - Platform Selection
      * Choose an execution platform for the system (i.e. hardware and the real-time operating system to be used)
        + Factor on choosing platform

Timing constraints

Power limitations

Experience of the development team

Price target for the product

* + - Stimuli/Response identification
      * Identifying the stimuli that the system must process and the associated response or responses for each
    - Timing Analysis
      * For each stimulus and associated response, you identify the timing constraints that apply to both stimulus and response processing. These are used to establish the deadlines for the processes in the system.
    - Process Design
      * You aggregate the stimulus and response processing into a number of concurrent processes.
      * A good starting point for designing the process architecture is the architectural patterns that are described in section 20.2
      * You then optimize the process architecture to reflect the specific requirements that you have to implement.
    - Algorithm Design
      * For each stimulus and response, design the algorithms to carry out the required computations
      * These have to be developed relatively early in the design process to give an indication of the amount of processing required and the time needed to complete that processing. For example, signal processing takes a lot of computation
    - Data Design
      * Specify the information that is exchanged by processes and the events that coordinate information exchange, and design data structures to manage this information exchange. Several concurrent processes may share these data structures
    - Process Scheduling
      * You design a scheduling system that will ensure that processes are started in time to meet their deadlines
    - \*\*\*The order of these may change depending on the product
* Processes in a real-time system have to be coordinated and share information.
  + Process coordination mechanisms ensure mutual exclusion (semaphores, monitors, critical regions, etc.) to shared resources
  + When design the information exchange between processes, you have to take into account the fact that these processes may be running at different speeds
    - Ex. One process produces info, another process consumes the info
      * If the producer is running faster than the consumer, new information could overwrite a previously read information item before the consumer process has read the original info.
      * If the consumer is running faster, the same item could be read twice
    - To solve this, you should implement information exchange using a shared buffer and use mutual exclusion mechanisms to control access to that buffer
      * This means that information can’t be overwritten before it has been read and that information cannot be read twice
      * Shared buffer are usually implemented as circular buffers (see page 544)
        + Important for produce and consumer to not access the same item at the same time. Also ensure the producer does not add items when the buffer is full.
        + Synchronization structures like mutexes ensure the Get() and Put() operations are synchronized -> they don’t access the same location at the same time. If the buffer is full, Put() has to wait; if the buffer is empty, the Get() process has to wait until an entry has been made
* Once you have chosen the execution platform, design process architecture, and decided on a scheduling policy, you may need to check that the system will meet its timing requirements.
  + You can do this:
    - Through static analysis of the system using knowledge of the timing behavior of components
    - Through simulation
  + These checks may reveal what needs to be redesigned to improve system performance
* Timing constraints or other requirements may sometimes mean that it is best to implement some system function like signal, processing in hardware.
* **Real-Time System Modeling (Section 20.1.1)**
  + Events that a real-time system must react to often cause the system to move from one state to another. For this reason, state models, are often used to describe real-time systems.
    - A state model of a system assumes that, at any time, the system is in one of a number of possible states. When a stimulus is received, this may cause a transition to a different state.
    - State models are language independent
* **Real-Time Programming (Section 20.1.2)**
  + Programming languages for real-time systems development have to include facilities to access system hardware, and it should be possible to predict the timing of particular operations in these languages.
  + Hard real-time systems are still sometime developed in assembly language. Systems-level languages like C are widely used.
  + Advantages of using systems programming languages like C:
    - It allows the development of very efficient programs
  + Disadvantages of using systems programming languages like C:
    - Do not include constructs to support concurrency or the management of shared resources. To use these, you must go through the operating system. Calls to these functions cannot be checked by the compiler, so programming errors are more likely.
    - Programs are more difficult to understand because the language does not include real-time features.
    - Reader has to know how real-time support is provided using system calls
  + Because of timing constraints, you may not be able to use object-oriented development for hard real-time systems
    - OOD involves hiding data representations and accessing attribute values through operation define within the object. This means that there is a significant performance overhead in object-oriented systems because extra code is required to mediate access to attributes and handle calls to operations. This loss of performance may make it impossible to meet real-time deadlines

**Architectural Patterns (Section 20.2)**

* These are abstract descriptions of good design practice. They encapsulate knowledge about the organization of system architectures, when these architectures should be used and their advantages and disadvantages.
* Unlike regular SW architectural patterns, embedded system architectural patterns are process-oriented rather than object- or component-oriented.
* 3 Patterns
  + Observe and React
    - Used when a set of sensors are routinely monitored and displayed. When the sensors show that some event has occurred, the system reacts by initiating a process to handle the event.
  + Environmental Control
    - Used when a system includes sensors, which provide info about the environment and actuators that can change the environment. In response to environmental changes detected by the sensor, control signals are sent to the system actuators
  + Process Pipeline
    - Used when data has to be transformed from one representation to another before it can be processed. The transformation can be implemented as a sequence of processing steps, which may be carried out concurrently. This allows for very fast data processing, because a spate core or processor can execute each transformation.
* **Observe and React (Section 20.2.1)**
  + Monitoring systems are an important class of embedded real-time systems.
  + A monitoring system examines its environment through a set of sensors and, usually, displays the state of the environment in some way.
    - If some exceptional event or sensor state is detected by the system the monitoring system takes some action. Often, this involves raising an alarm to draw an operator’s attention to the event.
    - Observe and React pattern is used commonly in monitoring systems
  + Patter Info
    - Description: The input values of a set of sensors of the same types are collected and analyzed. These values are displayed in some way. If the sensor values indicate that some exceptional condition has arisen, then actions are initiated to draw the operator’s attention to that value and, in certain cases, to take actions in response to the exceptional value.
    - Stimuli: Values from sensors attached to the system
    - Responses: Outputs to display, alarm triggers, signals to reacting systems
    - Processes: Observer, Analysis, Display, Alarm, Reactor
    - Used In: Monitoring system, alarm systems
* **Environment Control (Section 20.2.2)**
  + The most widespread use of embedded software is in control systems. In these systems, the software control the operation of equipment, based on stimuli from the equipment.
    - Example. ABS in cars controls brakes in case of tire skid
  + Pattern Info
    - Description: The system analyzes info from a set of sensors that collect data from the environment. Further info may also be collected on the state of the actuators that are connected to the system. Based on the data from the sensors and actuators, control signals are sent to the actuators that then cause changes to the system’s environment. Info about the sensor values and the state of the actuators may be displayed.
    - Stimuli: Values from sensors attached to the system and the state of the system actuators
    - Responses: Control signals to actuators, display info
    - Processes: Monitor, control, display, actuator driver, actuator monitor
* **Process Pipeline (Section 20.2.3)**
  + Many real-time systems are concerned with collecting data from the system’s environment, then transforming that data from its original representation into some other digital representation that can be more readily analyzed and processed by the system. The system may also convert digital data to analog data, which it then sends to its environment
    - Ex. SW radio accepts incoming packets of digital data representing the radio transmission and transforms these into a sound signal that people can list to
  + Data processing that is involved in many of these systems has to be carried out very quickly; or else the incoming data may be lost and outgoing signals may be broken up because essential information is missing.
  + This pattern breaks down the required data processing into sequence of separate transformations, with each transformation carried out by an independent process.
  + Pattern Info
    - Description: A pipeline of processes is set up with data moving in sequence from one end of the pipeline to another. The processes are often linked by synchronized buffers to allow the producer and consumer processes to run at different speeds. The culmination of a pipeline may by display or data storage or the pipeline may terminate in an actuator.
    - Stimuli: Input values from the environment or some other process
    - Responses: Output values to the environment of shared buffer
    - Processes: Producer, buffer, and consumer
    - Used in: Data acquisition systems, multimedia systems

**Timing Analysis (Section 20.3)**

* The correctness of a real-time system depends not just on the correctness of its outputs but also on the time at which these outputs were produced. This means that an important activity in the embedded, real-time software development process is timing analysis
* In timing analysis, you calculate how often each process in the system must be executed to ensure that all inputs are processed and all system responses, are produced in a timely way. The results of the timing analysis are used to decide how frequently each process should execute and how these processes should be scheduled by the RTOS.
* Timing analysis is difficult for RTOSs that must deal with a mixture of periodic and aperiodic stimuli.
  + Because aperiodic stimuli are unpredictable, you make assumptions about the probability of these stimuli occurring and therefore requiring service at any particular time. These assumptions may be incorrect and system performance after delivery may not be adequate.
* When analyzing the timing requirements of embedded real-time systems and designing systems to meet these requirements, there are 3 key factors that you have to consider:
  + Deadlines
    - The times by which stimuli must be processed and some response produced by the systems
    - If the system does not meet a deadline then, if it is a hard RTOS, it is a system failure; if it is a soft RTOS, it results in degraded system quality
  + Frequency
    - The number of times per second that a process must execute so that you are confident that it can always meets its deadlines
  + Execution Time
    - Time required to process a stimulus and produce a response.
    - You have to keep track of the average execution time and the worst execution time
* After creating a list of timing constraints for each class of sensor separately, you should map the system functions to concurrent processes. When you have completed the timing analysis, you may then annotate the process model with information about frequency of execution and their expected execution time.
* Final step in the design process is to design a scheduling system that will ensure that a process will always be scheduled to meet its deadlines. You can only do this if you know the scheduling approaches that are supported by the RTOS.
  + The RTOS allocates a process to a processor for a given amount of time. The time can be fixed, or may vary depending on the priority of the process. In allocating process priorities, you have to consider the deadlines of each process so that processes with short deadlines receive processor time to meet these deadlines.

**Real-Time Operating Systems (Section 20.4)**

* The execution platform for most application systems is an operating system that manages shared resources and provides features such as a file system, run-time process management, etc. However, the extensive functionality in a conventional OS takes up a great deal of space and slows down the operation of programs. Also, the process management features in the system may not be designed to allow fine-grain control over the scheduling or processes.
  + Because of this Linux and Windows OS are not normally used in real-time systems
  + Very simple embedded system may be implemented as “bare metal” systems. The system themselves include system startup and shutdown, process and resource management, and process scheduling
    - More commonly, embedded systems are built on top of a RTOS
* At the bare minimum, RTOSs usually include
  + A real-time clock, which provides the information required to schedule processes periodically
  + An interrupt handler, which manages aperiodic requests for service
  + A scheduler, which is responsible for examining the processes that can be executed and choosing one of these for execution
  + A resource manager, which allocates appropriate memory and processor resources to processes that have been scheduled for execution
  + A dispatcher, which is responsible for starting the execution of processes
* **Process Management (Section 20.4.1)**
  + Real time systems have to handle external events quickly and, in some cases, meet deadlines for processing these events. This means that the event-handling processes must be scheduled for execution in time to detect the event. They must also be allocated sufficient processor resources to meet their deadline. The process manager in an RTOS is responsible for choosing process for execution, allocating processor and memory resources, and starting and stopping process execution on a processor.
  + Process manager has to manage processes with different priorities. For some stimuli it is essential that their processing should be completed withint he specified time limis. Other processes may be safely delayed if a more critical process requires service. Consequently, the RTOS has to be able to manage at least 2 priority levels for system processes:
    - Interrupt level
      * Is the highest priority level. It is allocated to processes that need a very fast response. Once of these processes will be the real-time clock process
    - Clock Level
      * Level of priority is allocated to periodic processes
  + Within each of these priority levels, different classes of process may be allocated with different priorities.
    - Deciding the priority of a process takes extensive analysis and simulation
  + Periodic processes are processes that must be executed at specified time intervals for data acquisition and actuator control.
    - Using timing requirements specified in the application program, the RTOS arranges the execution of periodic processes so that they can all meet their deadlines
  + Processes that have to respond quickly to asynchronous events may be interrupt-driven. The computer’s interrupt mechanism causes control to transfer to a pre-determined memory location. This location contains an instruction to jump to a simple and fast interrupt service routine. The service routine disables further interrupts to avoid being interrupted itself. It then discovers the cause of the interrupt and initiates, with a high priority, a process to handle the stimulus causing the interrupt.
  + 2 commonly used scheduling strategies:
    - Non-preemptive scheduling
      * Once a process has been scheduled for execution it runs to completion or until it is blocked for some reason, such as waiting for input.
      * Negatives are that high priority processes have to wait for low priority processes to finish
    - Preemptive Scheduling
      * Execution of an executing process may be stopped if a higher priority process requires service. The higher priority process preempts the execution of the lower priority process and is allocated to a processor.
    - These either use round-robin scheduling, highest-frequency process first, and shortest dealing first scheduling.
  + Information about the process to be executed is passed to the resource manager. The RM allocates memory, and in a multiprocessor system, also adds a processor to this process. The process is then placed on the ready queue (list of processes ready for execution). When a processor finishes executing a process and becomes available, the dispatcher is invoked. It scans the ready list to find a process that can be executed on the available processor and starts its execution.