

UNIT-III PROCESSES AND OPERATING SYSTEMS

Structure of a real – time system – Task Assignment and Scheduling – Multiple Tasks and Multiple Processes – Multirate Systems – Pre-emptive real – time Operating systems – Priority based scheduling – Interprocess Communication Mechanisms – Distributed Embedded Systems – MPSoCs and Shared Memory Multiprocessors – Design Example – Audio Player, Engine Control Unit and Video Accelerator.

2 Marks questions with answers

1. Define process. (Nov/Dec 2013)

A process is defined as a single execution of a program. A running of the same program two different times leads to creation of two different processes. Each process has its own state that includes its register and memory.

2. What are the states of a process?

The states of a process are

- a. Running
- b. Ready
- c. Waiting

3. What is the function in ready state?

Processes which are ready to run but are not currently using the processor are said to be in the 'ready' state.

4. Define scheduling.

Scheduling is defined as the work of choosing the order of running process. It is a process of selection which says that a process has the right to use the processor at given time.

5. What is scheduling policy?

Scheduling policy describes the way in which processes are chosen to get promotion from ready state to running state.

6. Define hyperperiod.

Hyperperiod is the least-common multiple of the periods of all the processes. It is a finite period that covers all possible combinations of process executions.

7. What is schedulability?

Schedulability indicates, any execution schedule is there for a collection of process in the system's functionality.

8. What are the types of scheduling? (May/June 2013)

The types of scheduling are

1. Time division multiple access scheduling.
2. Round robin scheduling.
3. Cyclostatic scheduling.

9. What is cyclostatic scheduling?

Cyclostatic scheduling is a type of scheduling in which schedules are separated into equal sized time slots.

10. Define round robin scheduling.

Round robin scheduling is a type of scheduling which employs the hyperperiod as an interval. In this scheduling, the processes run in the given order.

11. What is scheduling overhead?

Scheduling overhead is the time of execution needed to select the next execution process.

12. What is meant by context switching? (April 2014), (Nov/Dec 2013), (Nov/Dec 2012) (April 2018)

- A context switch is the computing process of storing and restoring of a CPU, where execution can be resumed from the same point at a later time.
- The context switching is an essential feature of multitasking operations system.

13. Define priority scheduling.

Priority scheduling is defined as a scheduling which maintains a priority queue of processes that are in the runnable state.

14. What is rate monotonic scheduling?

Rate monotonic scheduling is an approach that is used to assign task priority for a preemptive system.

15. What is critical instant?

Critical instant is the situation in which the process or task possesses "highest response time."

16. What is critical instant analysis?

Critical instant analysis is used to know about the schedule of a system. It says that, based on the periods given, the priorities to the processes has to be assigned.

17. Define earliest deadline first scheduling.

Earliest deadline first scheduling is a type of scheduling which applies task priority policy that uses the nearest deadline as the criterion for assigning the task priority.

18. What is use of IDC mechanism?

IDC (Inter Domain Communication) mechanism is necessary for a process to get communicate with other process in order to attain a specific application in an operating system.

19. What are the two types of communication?

The two types of communication are

1. Blocking communication
2. Non-blocking communication

20. Give the different styles of inter process communication. (April 2014)

The different styles of inter process communication are

1. Shared memory communication.
2. Message passing.

21. What is ISR? (Nov 2013)

An interrupt handler, also known as an interrupt service routine (ISR). It is a callback subroutine in an operating system. It is a device driver whose execution is triggered by the reception of an interrupt.

22. List the uses of interrupt service routines? (May/June 2012)

The uses of interrupt service routines are

- Input/output data transfer for peripheral devices.
- Input signals to be used for timing purpose.
- Real-time executives/multitasking.
- Event-driven program.

23. What is meant by masking? (Nov 2013)

The priority mechanism must ensure that a lower-priority interrupt does not occur, when a higher-priority interrupt is being handled. The decision process is known as *masking*.

24. Write about non-maskable interrupt.

The highest-priority interrupt is normally called as *Non-Maskable Interrupt (NMI)*. The NMI cannot be turned off. It is usually reserved for interrupts caused by power failures. The NMI can be used to save critical state in nonvolatile memory. It turns off I/O devices to eliminate spurious device operation during power down.

25. What are the three conditions that must be satisfied by the re-entrant function? (May/June 2012, NOV 2017)

A function is called as a re-entrant function, when the following three conditions are satisfied,

- All the arguments pass the values and some of the argument is a pointer whenever a calling function calls it.
- When an operation is not atomic, the function should not operate on any variable, which is declared but passed by reference not passed by arguments in the function.
- That function does not call any other function that is not itself re-entrant.

26. Define operating system.

An operating system is defined as a program responsible for scheduling the CPU and controlling access to devices. Operating system is interface between user of a computer and computer hardware.

27. List out the common tasks of operating system.

The common tasks of operating system are

- Process management
- Memory management
- Device management
- Application management
- User interface

28. What is the power optimization strategies used for processes? (May/June 2013)

The power optimization strategies used for processes are

- Predictive shutdown
- Advanced Configuration and Power Interface.

29. What does a scheduler do in an operating system environment? (Nov/Dec 2012)

In an operating system, the scheduling process is carried out by software called scheduler.

30. List out the types of scheduler.

The types of scheduler are:

- Short term scheduler
- Mid-term scheduler
- Long term scheduler.

31. What is Task?

A task is an independent process that takes control of the CPU, when scheduled at an OS. Every task has a TCB (Task Control Block).

32. What is Task State?

A state of a task that changes on scheduler directions. A task at an instance can be one of the four states such as idle, ready, blocked and running. It is controlled by the scheduler.

33. Define interprocess communication.

An output from one task (or process) passed to another task through the scheduler. It uses signals, exceptions, semaphores, queues, mailboxes, pipes, sockets and remote procedure call. This is known as inter process communication.

34. Define Semaphore.

Semaphore is a special variable or function that is used to take note of certain actions to prevent another task or process from proceeding.

35. What is shared data problem?

If a variable is used in two different processes (tasks) and another task, if it interrupts without before the operation on that is completed, then the shared data problem arises.

36. What is priority inversion problem? How it can be solved?

A problem in which a low priority task does not release the process for a higher priority task. An operating system can take care of this by appropriate provisions.

This problem can be solved temporarily by boosting the low priority task to higher priority task.

This is called as priority inheritance.

37. Briefly discuss about Deadlock situation.

Deadlock is a situation in which the processes wait for the other resource which is occupied by another process in a loop. For example take processes P1 and P2 and resources

R1 and R2 in the deadlock condition. P1 -

R1 waits for R2

P2 - R2 waits for R1 so both processes wait for the other resource to get free for their complete operation.

38. What is Message Queue?

A task sending the multiple FIFO or priority message into a queue for use by another task using queue message as an input is said to be message queue.

39. What is Socket?

Socket provides the logical link. It is using a protocol between the tasks in a client server or peer-to-peer environment.

40. What is Remote Procedure Call?

Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network's details. A procedure call is also sometimes known as a function call or a subroutine call.

41. Define Thread.

Thread is a minimum unit for a scheduler to schedule the CPU and other system resources. A process may consist of multiple threads. A thread has an independent process control block like a task control block. A thread also executes codes under the control of a scheduler.

42. Give the uses of Task Control Block (TCB).

Task control block is a memory block which holds information of program counter, memory map, the signal (message) dispatch table, signal mask, task ID, CPU state (registers etc.) and a kernel stack (for executing system calls, etc.).

43. What are the problems that may arise while using semaphores?

The problems that may arise while using semaphores are,

- (i) Sharing of two semaphores creates a deadlock problem.
- (ii) Without a timeout an ISR worst-case latency may exceed the deadline.
- (iii) A semaphore not taken, and another task uses a shared variable.
- (iv) When using multiple semaphores, if an unintended task takes the semaphore, it creates a problem.
- (v) It may introduce priority inversion problem.

44. What are multi rate systems? and give two examples (April/May 2023)

Multirate embedded computing systems are very common, including automobile engines, printers and cell phones. In all these systems, certain operations must be executed periodically, and each operation is executed at its own rate. Examples are automobile engines, printers and cell phones.

45. What is release time and deadline of a process?

The release time is the time at which the process becomes ready to execute. A deadline specifies when a computation must be finished. The deadline for an aperiodic process is generally measured from the release time, since that is the only reasonable time reference.

46. What is the period and rate of a process?

The period of a process is the time between successive executions. The process's rate is the inverse of its period.

47. What is a task graph?

A set of processes with data dependencies is called a task graph.

48. Define CPU utilization.

Utilization is the ratio of the CPU time that is being used for useful computations to the total available CPU time. This ratio ranges between 0 and 1, with 1 meaning that all of the available CPU time is being used for system purposes.

49. What is the response time of a process?

The response time of a process is the time at which the process finishes its execution.

50. Define preemption.

Preemption is the act of forcing a process out of execution i.e. making a context switch.

51. If your set of processes is un-schedulable and you need to guarantee that they complete their deadlines, give possible ways to solve this problem?

The techniques available are as follows

- a. Get a faster CPU.
- b. Redesign the processes to take less execution time
- c. Rewrite the specification to change the deadlines

52. What are the benefits of multithreaded programming?

The benefits of multithreaded programming can be broken down into four major categories:

- Responsiveness
- Resource sharing
- Economy
- Utilization of multiprocessor architectures

53. What is a Dispatcher?

The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves:

- Switching context
- Switching to user mode
- Jumping to the proper location in the user program to restart that program.

54. What is dispatch latency?

The time taken by the dispatcher to stop one process and start another running is known as dispatch latency.

55. What are the various scheduling criteria for CPU scheduling?

The various scheduling criteria are

- CPU utilization
- Throughput
- Turnaround time
- Waiting time
- Response time

56. Define throughput.

Throughput in CPU scheduling is the number of processes that are completed per unit time. For long processes, this rate may be one process per hour. For short transactions, throughput might be 10 processes per second.

57. What is turnaround time?

Turnaround time is the interval from the time of submission to the time of completion of a process. It is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

58. What is a mailbox?

A message or message pointer from a task that is addressed to another task is said to be a mailbox.

59. State the model of Rate Monotonic Scheduling or What is Rate Monotonic Scheduling? (NOV/DEC 2018)

This simple model of the system is as follows,

- All processes run periodically on a single CPU.
- Context switching time is ignored.
- There are no data dependencies between processes.
- The execution time for a process is constant.
- All deadlines are at the ends of their periods.
- The highest-priority ready process is always selected for execution.

60. Write about POSIX (NOV 2017)

- It is a version of UNIX
- Serves as a good model for RTOS
- It has less options

61. Write advantages and limitations of priority scheduling.

Advantage: Very efficient in different situations
Limitation:

Very static scheduling

62. What is the concept of multitasking? What does it signify? (NOV/DEC 2018) (April/May 2019)

Multitasking, in an operating system, is allowing a user to perform more than one computer task at a time. The OS is able to track of where you are in these tasks and go from one to other without losing information.

63. What is priority inheritance and priority inversion? (April/May 2019)

In a line, *Priority Inversion* is a **problem** while *Priority Inheritance* is a **solution**.

Literally, *Priority Inversion* means that priority of tasks gets inverted and *Priority Inheritance* means that priority of tasks get inherited. Both of these phenomena happen in priority scheduling.

64. Mention the power saving strategy adopted for real-time systems (April/May 2019)

There are so many methods available to save the power in real-time systems. However, the main power saving is achieved through dynamic voltage scaling.

65. Define audio player.

Audio players are defined as any media player which can only play audio files. Players capable of video playback are included under the comparison of video player software, even if they are primarily well known for audio playback.

66. What are the basic functions of audio players? Or List out the major components of audio player (April/May 2019) (Dec 2022/Jan 2023)

An MP3 player performs three basic functions such as

1. Audio storage
2. Audio decompression
3. User interface

67. Define video accelerator or need for video accelerator (NOV 2017) (April/May 2019)

Video accelerator is a hardware circuit on a display adapter that speeds up full motion video, which also frees the CPU to take care of other tasks.

68. Define engine control unit.

An engine control unit is a type of electronic control unit that controls a series of accelerators on an internal combustion engine to ensure optimal engine performance.

69. Give one challenge in developing codes for MPSoCs (Dec 2022/Jan 2023)

MPSoCs present unique challenges for software development due to the change from uniprocessor to many or multiprocessors (implying parallelism) and non-uniform memory structures.

The programming model of all languages such as C, C++ etc. assumes a homogeneous implementation architecture with a uniform, shared memory space. This model is incompatible with the application-specific, heterogeneous architectures of MPSoCs. This results in a mismatch between the programmer's conceptual model and the underlying implementation.

70. How does the ARM SAI instruction provide atomic executions?

(Dec 2022/ Jan 2023)

To support the atomic instructions added in the Armv8.1 architecture, CHI-B provides Atomic Transactions. An interconnect uses Atomic Transactions to transport an atomic operation and its operands from one device to another. Using atomics instead of exclusive access reduces the amount of time during which data is inaccessible to other agents. Atomic Transactions can execute several atomic operations and can be performed internally or externally to the processor.

71. What is meant by requirement analysis if doing memory scaling for a video accelerator? (Dec 2022/Jan 2023)

The arithmetic on each pixel is simple, but we have to process a lot of pixels. If MBSIZE is 16 and SEARCHSIZE is 8, and remembering that the search distance in each dimension is $8 + 1 + 8$, then we must perform $N \text{ ops} = (16 \times 16) \times (17 \times 17) = 73,984$.

72. What are the scheduling states considered in a process? (April/May 2023)

The operating system considers a process to be in one of three basic scheduling states: **waiting**, **ready**, or **executing**. At most, there is one process executing on the CPU at any time. If there is no useful work to be done, an idling process may be used to perform null operations. Any process that could be executed is in the ready state; the operating system chooses among the ready processes to select the next executing process.

73. What is the significance of shared memory multiprocessors? (Nov/Dec 2023)

A shared memory multiprocessor is an architecture consisting of a modest number of processes, all of which have direct access to all the main memory in the system.

74. What is the dynamic priority algorithm? State its advantages and applications (Nov/Dec 2023)

The priorities of the process is changed on the fly if needed so that it meets the deadline. This kind of priority is very useful in real-time applications such as missile, aircraft, bullet train etc.

75. What are sporadic and aperiodic tasks? Give examples (Nov/Dec 2023)

The real-time task that reoccurs at any random instant and has hard deadlines is known as a sporadic real-time task. Example: fire handling task.

The dynamic tasks that reoccur at any random time and have soft deadlines are known as aperiodic, examples keyboard, mouse.

76. Define scheduling. (DEC 20, APR 21)

The first job of the OS is to determine that process runs next. The work of choosing the order of running processes is known as scheduling.

77. What are the performance measures of real-time system? (DEC 20, APR 21)

The following characteristics of real-time operating systems can be measured: **average task switch time, average pre-emption time, average interrupt latency, semaphore shuffle time, deadlock break time, and inter-task.**

78. Mention the limitations of RM algorithm. (Dec 21)

Deadlines must be similar to the time periods. Deadlines are deterministic. Process running with highest priority that needs to run, will preempt all the other processes.

79. State the control of aborting of illegal memory accesses help in fault tolerance (Dec 2022/Jan 2023)

The control of aborting of illegal memory accesses helps in fault tolerance. Each and every redundant memory block is protected such that there is no illegal memory access for program and data. Because keeping the redundant blocks secured is very essential in achieving fault tolerance. Moreover, the memory is a very important unit next to processor in an embedded system for overall performance.

80. What are the essential factors to be considered in estimating program run times? (April/May 2023)

- 1) Algorithm time complexity.
- 2) Input size.
- 3) CPU speed.
- 4) I/O waiting time.
- 5) Amount of running processes.

6) Amount of available memory.

7) Programming language used.

81. Mention the method of dealing with sporadic tasks (April/May 2023)

The real-time tasks that reoccur at any random instant and have hard deadline are known as sporadic real-time tasks. Basically all the high critical tasks are sporadic tasks. For example, fire handling task in industry or emergency message arrival in system are sporadic real-time tasks. It goes through acceptance test, executed only when sufficient slacktime is available and includes commands given by the system.

Question Bank: Realtime systems	
Most Important (90% to 100% possibility)	1. Structure of a Real Time System.
	2. Explain the basic model of real time systems.
	3. Explain Fault Tolerance Techniques.
(40% to 50% possibility)	4. Explain Clock Synchronization.
	5. Discuss about estimating program runtimes
	6. Explain Task Assignment and Scheduling

Question Bank: Process and operating systems	
Most Important (90% to 100% possibility)	1. Explain in detail about Inter-process communication mechanisms.
	2. Discuss in detail about Preemptive real time operating systems.
	3. Design of a video accelerator
	4. Briefly discuss about various power optimization strategies for processes.
(40% to 50% possibility)	5. Design of an Audio Player.
	6. Explain in detail the design of engine control unit.

PART-B

(Structure of a Real Time System Task Assignment and Scheduling.)

- Most operating systems (even those that support multiple scheduling policies) schedule all applications according to the same scheduling algorithm at any given time.

- Whether each application can meet its timing requirements is determined by a global schedulability analysis based on parameters of every task in the system.
- The necessity of detailed timing and resource usage information of all applications that may run together often forces the applications to be developed together and, thus, keeps the system closed.
- Hard real-time applications can run with soft real-time and non-real-time applications in this environment. It makes use of the two-level scheduling scheme.
- This scheme enables each real-time application to be scheduled in a way best suited for the application and the schedulability of the application to be determined independent of other applications that may run with it on the same hardware platform.

Structure of a Real Time System

Q1. Discuss about task and processes. Tas

ks and Processes

- Many (if not most) embedded computing systems do more than one thing—that is, the environment can cause mode changes that in turn cause the embedded system to behave quite differently.
- For example, when designing a telephone answering machine, one can define recording a phone call and operating the user's control panel as distinct tasks, because they perform logically distinct operations and they must be performed at very different rates.
- These different tasks are part of the system's functionality, but that application-level organization of functionality is often reflected in the structure of the program as well.

- A process is a single execution of a program. If a user runs the same program two different times, then two different processes are created. Each process has its own state that includes not only its registers but also its memory.
- In some OSs, the memory management unit is used to keep each process in a separate address space. In others, particularly lightweight RTOSs, the processes run in the same address space. Processes that share the same address space are often called threads. The terms tasks and processes are somewhat interchangeable, as do many people in the field. To be more precise, a task can be composed of several processes or threads; it is also true that a task is primarily an implementation concept and processes more of an implementation concept.
- To understand why the separation of an application into tasks may be reflected in the program structure, consider how to build a stand-alone compression unit based on the compression algorithm. This device is connected to serial ports on both ends.
- The input to the box is an uncompressed stream of bytes. The box emits a compressed string of bits on the output serial line, based on a predefined compression table. Such a box may be used, for example, to compress data being sent to a modem.
- The program's need to receive and send data at different rates—for example, the program may emit 2 bits for the first byte and then 7 bits for the second byte—will obviously find itself reflected in the structure of the code.
- It is easy to create irregular, ungainly code to solve this problem; a more elegant solution is to create a queue of output bits, with those bits being removed from the queue and sent to the serial port in 8-bit sets.
- But beyond the need to create a clean data structure that simplifies the control structure of the code, and it is necessary to ensure that input and output signals are to be processed at the proper rates.
- For example, if too much of time is spent in packaging and emitting output characters, then an input character can be dropped. Solving timing problems is a more challenging problem.

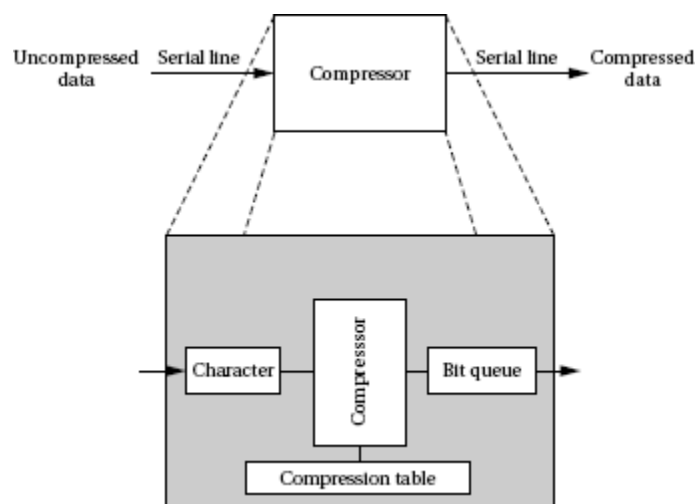


Figure 4.1 An on-the-fly compression box.

Major States. Our subsequent discussion focuses primarily on priority-driven systems. There are five major states of a thread.

- **Sleeping:** A periodic, aperiodic, or server thread is put in the sleeping state immediately after it is created and initialized. It is released and leaves the state upon the occurrence of an event of the specified types.
- Upon the completion of a thread that is to execute again, it is reinitialized and put in the sleeping state. A thread in this state is not eligible for execution.
- **Ready:** A thread enters the ready state after it is released or when it is preempted. A thread in this state is in the ready queue and is eligible for execution.
- **Executing:** A thread is in the executing state when it executes.
- **Suspended (or Blocked):** A thread that has been released and is yet to complete enters the suspended (or blocked) state when its execution cannot proceed for some reason. The kernel puts a suspended thread in the suspended queue.
- **Terminated:** A thread that will not execute again enters the terminated state when it completes. A terminated thread may be destroyed.

4.1.2. The Kernel

Q2. Explain the structure of Microkernel.

- Again, with a few exceptions, a real-time operating system consists of a microkernel that provides the basic operating system functions described below. Figure 4.2 shows a general structure of a microkernel.
- There are three reasons for the kernel to take control from the executing thread and execute itself: to respond to a system call, do scheduling and service timers, and handle external interrupts. The kernel also deals with recovery from hardware and software exceptions, but activities are ignored.
- **System Calls.** The kernel provides many functions which, when called, do some work on behalf of the calling thread. An application can access kernel data and code via these functions. They are called Application Program Interface (API) functions.
- A system call is a call to one of the API functions. In a system that provides memory protection, user and kernel threads execute in separate memory spaces.
- Upon receiving a system call, the kernel saves the context of the calling thread and switches from the user mode to the kernel mode. It then picks up the function name and arguments of the call from the thread's stack and executes the function on behalf of the thread.
- When the system call completes, the kernel executes a return from exception. As a result, the system returns to the user mode. The calling thread resumes if it still has the highest priority. If the system call causes some other thread to have the highest priority, then that thread executes.
- The calling thread is blocked until the kernel completes the called function. When the call is asynchronous (e.g., in the case of an asynchronous I/O request), the calling thread

continue to execute after making the call. The kernel provides a separate thread to execute the called function.

- Many embedded operating systems do not provide memory protection; the kernel and user execute in the same space.
- Reasons for this choice are the relative trustworthiness of embedded applications and the need to keep overhead small. (The extra memory space needed to provide full memory protection is on the order of a few kilobytes per process. This overhead is more serious for small embedded applications than the higher context-switch overhead that also incurs with memory protection.)
- In such a system, a system call is just like a procedure or function call within the application. Figure 4.2 shows examples of thread management functions: create thread, suspend thread, resume thread and destroy thread.
- The timer functions listed below them exemplify the time services a real-time operating system provides.

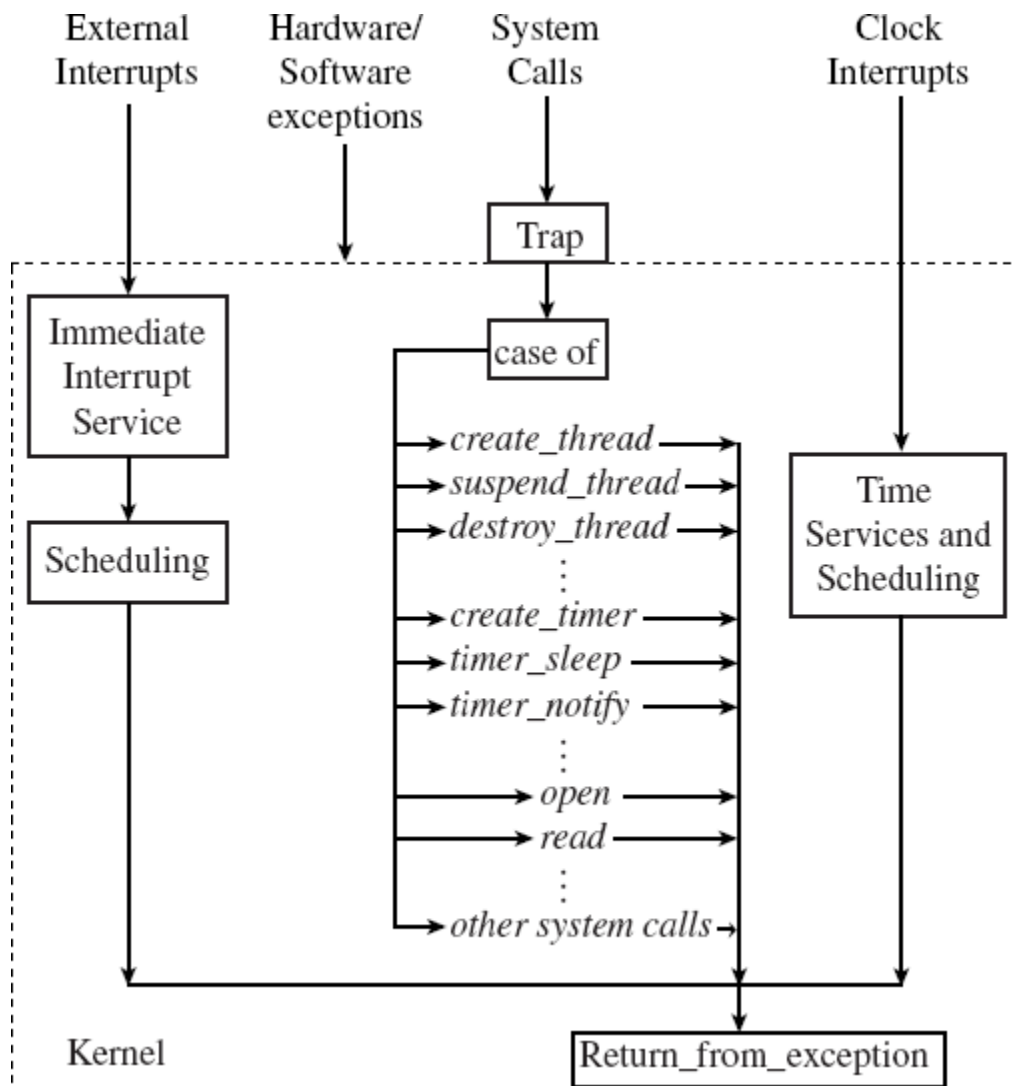


Figure 4.2 Structure of a microkernel. 4.1.3. A Basic Model of a Real-Time System

Q3. Explain the basic model of real-time systems (DEC20, APR21)

- The output interface, output conditioning, and the actuator are interfaced in a complementary manner. In the following, content briefly describes the roles of the different functional blocks of a real-time system.
- **Sensor:** A sensor converts some physical characteristic of its environment into electrical signals. An example of a sensor is a photo-voltaic cell which converts light energy into electrical energy. A wide variety of temperature and pressure sensors are also used.
- A temperature sensor typically operates based on the principle of a thermocouple. Temperature sensors based on many other physical principles also exist.
- For example, one type of temperature sensor employs the principle of variation of electrical resistance with temperature (called a varistor). A pressure sensor typically operates based on the piezoelectricity principle. Pressure sensors based on other physical principles also exist.
- **Actuator:** An actuator is any device that takes its inputs from the output interface of a computer and converts these electrical signals into some physical action on its environment.
- The physical actions may be in the form of motion, change of thermal, electrical, pneumatic, or physical characteristics of some objects. A popular actuator is a motor. Heaters are also very commonly used. Besides, several hydraulic and pneumatic actuators are also popular.
- **Signal Conditioning Units:** The electrical signals produced by a computer can rarely be used to directly drive an actuator. The computer signals usually need conditioning before they can be used by the actuator.
- This is termed output conditioning. Similarly, input conditioning is required to be carried out on sensor signals before they can be accepted by the computer.

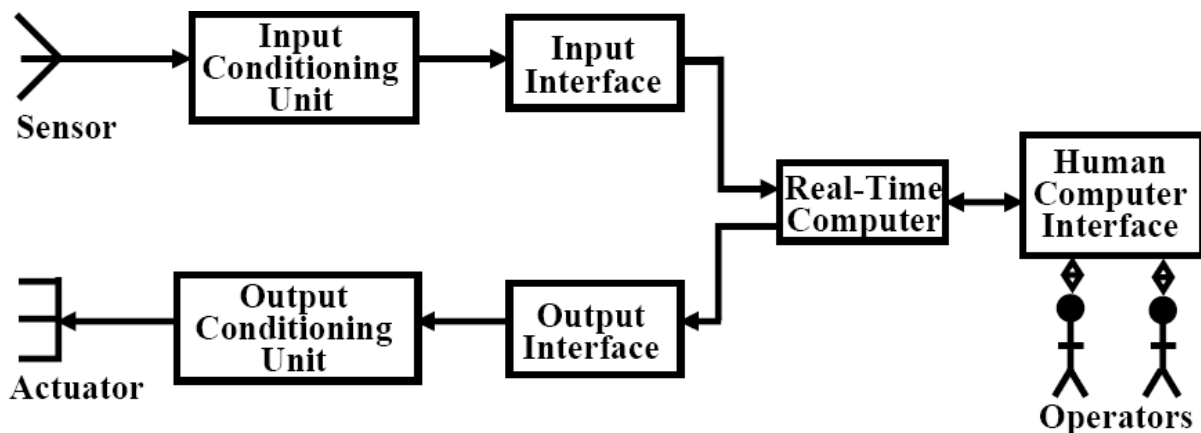


Figure 4.3 A model of real-time operating system

For example, analog signals generated by a photo-voltaic cell are normally in the milli-volts range and need to be conditioned before they can be processed by a computer.

- The following are some important types of conditioning carried out on raw signals

generated by sensors and digital signals generated by computers:

- Voltage Amplification: Voltage amplification is normally required to be carried out to match the full scale sensor voltage output with the full scale voltage input to the interface of a computer.
- For example, a sensor might produce voltage in the millivolts range, whereas the input interface of a computer may require the input signal level to be of the order of a volt.
- Voltage Level Shifting: Voltage level shifting is often required to align the voltage level generated by a sensor with that acceptable to the computer.
- For example, a sensor may produce voltage in the range -0.5 to +0.5 volt, whereas the input interface of the computer may accept voltage only in the range of 0 to 1 volt. In this case, the sensor voltage must undergo level shifting before it can be used by the computer.
- Frequency Range Shifting and Filtering: Frequency range shifting is often used to reduce the noise components in a signal. Many types of noise occur in narrow bands and the signal must be shifted from the noise bands so that noise can be filtered out.
- Signal Mode Conversion: A type of signal mode conversion that is frequently carried out during signal conditioning involves changing direct current into alternating current and vice-versa.
- Another type of signal mode conversion that is frequently used is conversion of analog signals to a constant amplitude pulse train such that the pulse rate or pulse width is proportional to the voltage level.
- Conversion of analog signals to a pulse train is often necessary for input to systems such as transformer coupled circuits that do not pass direct current. D/A

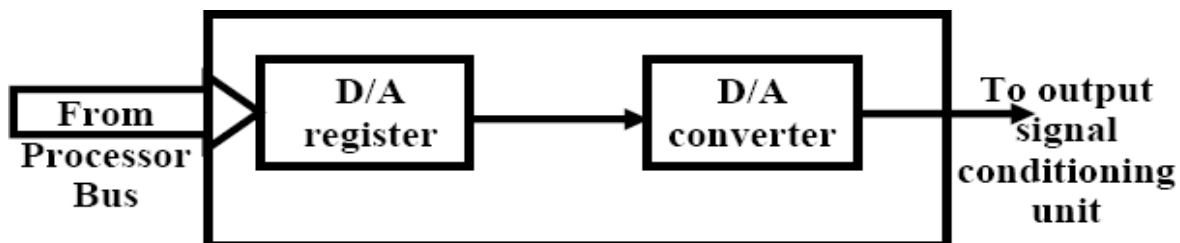


Figure 4.4 An output interface.

- Interface Unit: Normally commands from the CPU are delivered to the actuator through an output interface. An output interface converts the stored voltage into analog form and then outputs this to the actuator circuitry.

4.1.4. Estimating program runtimes

Q4. Discuss about estimating program run times / Enumerate the need for host based systems for stages of simulation, porting kernels and estimating program runtimes in embedded application deployment (DEC2022/JAN2023)

- Processes can have several different types of timing requirements imposed on them by the application. The timing requirements on a set of processes strongly influence the type of scheduling that is appropriate.
- A scheduling policy must define the timing requirements that it uses to determine whether a schedule is valid. Before studying scheduling properly, some outline of the types of process timing requirements that are useful in embedded system design.
- Figure 4.5 illustrates different ways in which two important requirements on processes can be defined: **release time** and **deadline**.
- The release time is the time at which the process becomes ready to execute; this is not necessarily the time at which it actually takes control of the CPU and starts to run.
- An aperiodic process is by definition initiated by an event, such as external data arriving or data computed by another process.
- The release time is generally measured from that event, although the system may want to make the process ready at some interval after the event itself.
- For a periodically executed process, there are two common possibilities. In simpler systems, the process may become ready at the beginning of the period. More sophisticated systems, such as those with data dependencies between processes, may set the release time at the arrival time of certain data, at a time after the start of the period.
- A deadline specifies when a computation must be finished. The deadline for an aperiodic process is generally measured from the release time, since that is the only reasonable time reference. The deadline for a periodic process may in general occur at some time other than the end of the period. Some scheduling policies make the simplifying assumption that the deadline occurs at the end of the period.

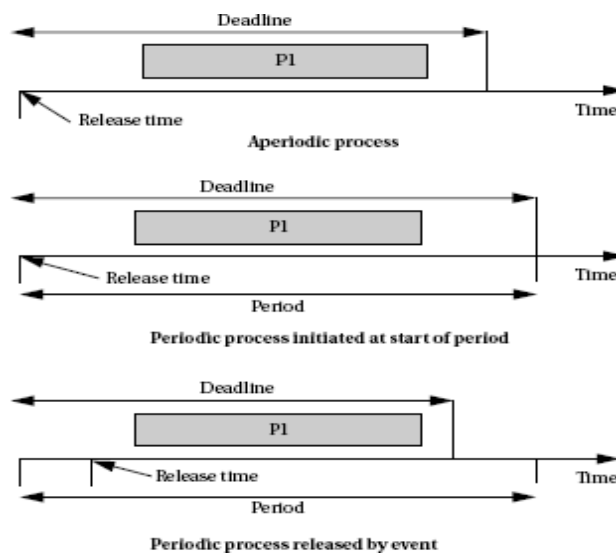


Figure 4.5 Example definitions of release times and deadlines.

- Rate requirements are also fairly common. A rate requirement specifies how quickly processes must be initiated.
- The **period** of a process is the time between successive executions. For example, the period of a digital filter is defined by the time interval between successive input samples.
- The process's **rate** is the inverse of its period. In a multi-rate system, each process executes at its own distinct rate.
- The most common case for periodic processes is for the initiation interval to be equal to the period. However, pipelined execution of processes allows the initiation interval to be less than the period. Figure 4.6 illustrates process execution in a system with four CPUs.
- The various execution instances of program P1 have been subscripted to distinguish their initiation times. In this case, the initiation interval is equal to one-fourth of the period.
- It is possible for a process to have an initiation rate less than the period even in single-CPU systems.
- If the process execution time is significantly less than the period, it may be possible to initiate multiple copies of a program at slightly offset times. Violation depends on the application—the results can be catastrophic in an automotive control system, whereas a missed deadline in a multimedia system may cause an audio or video glitch.
- The system can be designed to take a variety of actions when a deadline is missed. Safety-critical systems may try to take compensatory measures such as approximating data or switching into a special safety mode. Systems for which safety is not as important may take simple measures to avoid propagating bad data, such as inserting silence in a phone line, or may completely ignore the failure.
- Even if the modules are functionally correct, their timing or improper behavior can introduce major execution errors. Application Example 4.6 describes a timing problem in space shuttle software that caused the delay of the first launch of the shuttle.

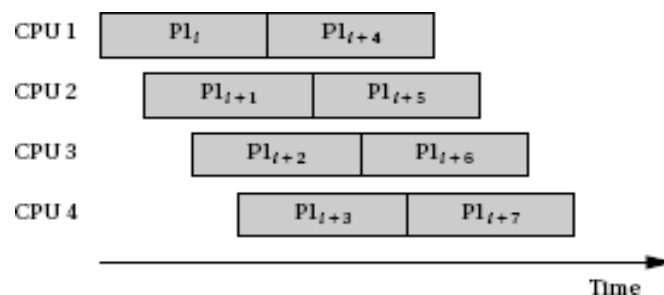


Figure 4.6 A sequence of processes with a high initiation rate.

- The order of execution of processes may be constrained when the processes pass data between each other. Figure 4.6 shows a set of processes with data dependencies among them. Before a process can become ready, all the processes on which it depends must complete and send their data to it.

- The data dependencies define a partial ordering on process execution—P1 and P2 can execute in any order (or in interleaved fashion) but must both complete before P3, and P3 must complete before P4. All processes must finish before the end of the period.
- The data dependencies must form a directed acyclic graph (DAG)—a cycle in the data dependencies is difficult to interpret in a periodically executed system.
- A set of processes with data dependencies is known as a **task graph**. Although the terminology for elements of a task graph varies from author to author, some of the components of the task graph is considered (a set of nodes connected by data dependencies) as a **task** and the complete graph as the **taskset**.
- The figure also shows a second task with two processes. The two tasks ($\{P1, P2, P3, P4\}$ and $\{P5, P6\}$) have no timing relationships between them.
- Communication among processes that run at different rates cannot be represented by data dependencies because there is no one-to-one relationship between data coming out of the source process and going into the destination process. Nevertheless, communication among processes of different rates is very common.
- Figure 4.7 illustrates the communication required among three elements of an MPEG audio/video decoder.
- Data come into the decoder in the system format, which multiplexes audio and video data. The system decoder process demultiplexes the audio and video data and distributes it to the appropriate processes.
- Multirate communication is necessarily one way—for example, the system process writes data to the video process, but a separate communication mechanism must be provided for communication from the video process back to the system process.

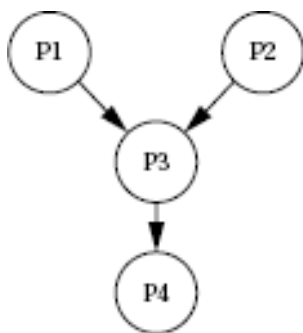


Figure 4.7 Data dependencies among processes

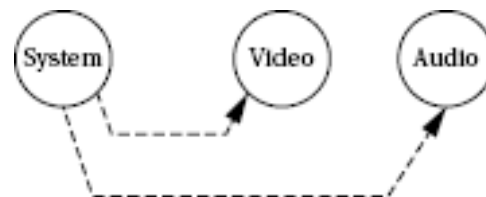


Figure 4.8 Communication among processes at different rates.

CPU usage metrics

A basic measure of the efficiency with usage of CPU. The simplest and most direct measure is utilization: Utilization = $\frac{\text{CPU time for useful work}}{\text{Total available CPU time}}$

CPU time for useful work

Total available CPU time

Utilization is the ratio of the CPU time that is being used for useful computations to the total available CPU time. This ratio ranges between 0 and 1, with 1 meaning that all of the available CPU time is being used for system purposes. The utilization is often expressed as a percentage.

4.1.5. Task Assignment and Scheduling

Q5. Explain Task Assignment and Scheduling / Scheduling of real-time systems / what is the purpose of priority-based scheduling? Discuss in detail with appropriate diagrams

(Dec 2022 / Jan 2023, May 2023, Dec 2023)

Scheduling

- The first job of the OS is to determine that process runs next. The work of choosing the order of running processes is known as scheduling.
- The OS considers a process to be one of three basic scheduling states: waiting, ready, or executing. There is at most one process executing on the CPU at any time. (If there is no useful work to be done, an idling process may be used to perform a null operation.) Any process that could execute is in the ready state; the OS chooses among the ready processes to select the next executing process.
- A process may not, however, always be ready to run. For instance, a process may be waiting for data from an I/O device or another process, or it may be set to run from a timer that has not yet expired. Such processes are in the waiting state.
- A process goes into the waiting state when it needs data that it has not yet received or when it has finished all its work for the current period.
- A process goes into the ready state when it receives its required data and when it enters a new period.
- A process can go into the executing state only when it has all its data, is ready to run, and the scheduler selects the process as the next process to run.

Scheduling Policies

- A scheduling policy defines how processes are selected for promotion from the ready state

to the running state. Every multitasking OS implements some type of scheduling policy.

- Choosing the right scheduling policy not only ensures that the system will meet all its timing requirements, but it also has a profound influence on the CPU horsepower required to implement the system's functionality.
- Schedulability means whether there exists a schedule of execution for the processes in a system that satisfies all their timing requirements. In general, it is necessary to construct a schedule to show schedulability, but in some cases some set of processes as unschedulable using some very simple tests can be eliminated.
- Utilization is one of the key metrics in evaluating a scheduling policy. Our most basic requirement is that CPU utilization be no more than 100% since one can't use the CPU more than 100% of the time.



Figure 4.9 Scheduling states of a process.

Priority-Based Scheduling

- After assigning priorities, the OS takes care of the rest by choosing the highest-priority ready process.
- There are two major ways to assign priorities: static priorities that do not change during execution and dynamic priorities that do change.

Rate-Monotonic Scheduling

- Rate-monotonic scheduling (RMS), introduced by Liu and Layland, was one of the first scheduling policies developed for real-time systems and is still very widely used. RMS is a static scheduling policy.
- It turns out that these fixed priorities are sufficient to efficiently schedule the processes in many situations. The theory underlying RMS is known as rate-monotonic analysis (RMA).
- This theory, as summarized below, uses a relatively simple model of the system.
 - All processes run periodically on a single CPU.
 - Context switching time is ignored.
 - There are no data dependencies between processes.
 - The execution time for a process is constant.

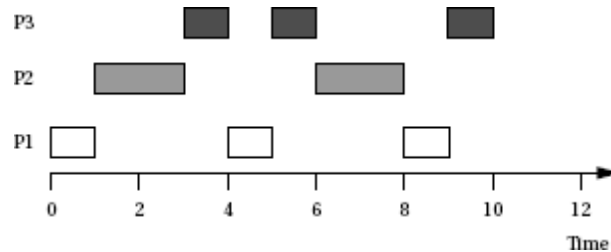
- All deadlines are at the ends of their periods.
- The highest-priority ready process is always selected for execution.
- The major result of RMA is that a relatively simple scheduling policy is optimal under certain conditions.
- Priorities are assigned by rank order of period, with the process with the shortest period being assigned the highest priority.
- This fixed-priority scheduling policy is the optimum assignment of static priorities to processes, in that it provides the highest CPU utilization while ensuring that all processes meet their deadlines.

4.1.6. Example: Rate-monotonic scheduling.

Here is a simple set of processes and their characteristics.

Process	Execution time	Period
P1	1	4
P2	2	6
P3	3	12

Applying the principles of RMA, P1 is provided with the highest priority, P2 the middle priority, and P3 the lowest priority. To understand all the interactions between the periods, it is needed to construct a timeline equal in length to hyperperiod, which are 12 in this case.



- All three periods start at time zero. P1's data arrive first. Since P1 is the highest-priority process, it can start to execute immediately.
- After one time unit, P1 finishes and goes out of the ready state until the start of its next period. At time 1, P2 starts executing as the highest-priority ready process. At time 3, P2 finishes and P3 starts executing. P1's next iteration starts at time 4, at which point it interrupts P3. P3 gets one more time unit of execution between the second iterations of P1 and P2, but P3 does not get to finish until after the third iteration of P1.
- Consider the following different set of execution times for these processes, keeping the same deadlines.
- In this case, no feasible assignment of priorities that guarantees scheduling can be shown. Even though each process alone has an execution time significantly less than its period, combinations of processes can require more than 100% of the available CPU cycles.

- For example, during one 12 time-unit interval, P1 was executed three times, requiring 6 units of CPU time; P2 twice, costing 6 units of CPU time; and P3 one time, requiring 3 units of CPU time. The total of $6 + 6 + 3 = 15$ units of CPU time is more than the 12 time units available, clearly exceeding the available CPU capacity.

Process	Execution time	Period
P1	2	4
P2	3	6
P3	3	12

4.1.7. Earliest-Deadline-First Scheduling

- Earliest deadline first (EDF) is another well-known scheduling policy that was also studied by Liu and Layland. It is a dynamic priority scheme—it changes process priorities during execution based on initiation times. As a result, it can achieve higher CPU utilization than RMS.
- The EDF policy is also very simple: It assigns priorities in order of deadline. The highest-priority process is the one whose deadline is nearest in time, and the lowest priority process is the one whose deadline is farthest away.
- Clearly, priorities must be recalculated at every completion of a process. However, the final step of the OS during the scheduling procedure is the same as for RMS—the highest-priority ready process is chosen for execution.
- The implementation of EDF is more complex than the RMS code. The major problem is keeping the processes sorted by time to deadline—since the times to deadlines for the processes change during execution.
- To avoid resorting the entire set of records at every change, a binary tree to keep the sorted records can be built and incrementally update the sort.
- At the end of each period, the records are moved to its new place in the sorted list by deleting it from the tree and then adding it back to the tree using standard tree manipulation techniques.
- And process priorities by traversing them in sorted order need to be updated, so the incremental sorting routines must also update the linked list pointers that let us traverse the records in deadline order. (The linked list lets us avoid traversing the tree to go from one node to another, which would require more time.)
- After putting in the effort to building the sorted list of records, selecting the next executing process is done in a manner similar to that of RMS. However, the dynamic sorting adds complexity to the entire scheduling process.
- Each update of the sorted list requires $O(\log n)$ steps. The EDF code is also significantly more complex than the RMS code.

- Which scheduling policy is better: RMS or EDF? That depends on criteria of a user. EDF can extract higher utilization out of the CPU, but it may be difficult to diagnose the possibility of an imminent overload.
- Because the scheduler does take some overhead to make scheduling decisions, a factor that is ignored in the schedulability analysis of both EDF and RMS, running a scheduler at very high utilization is somewhat problematic.
- RMS achieves lower CPU utilization but is easier to ensure that all deadlines will be satisfied. In some applications, it may be acceptable for some processes to occasionally miss deadlines. For example, a set-top box for video decoding is not a safety-critical application, and the occasional display artifacts caused by missing deadlines may be acceptable in some markets.

What if your set of processes is unschedulable and you need to guarantee that they complete their deadlines? There are several possible ways to solve this problem:

- Get a faster CPU. That will reduce execution times without changing the periods, giving you lower utilization. This will require you to redesign the hardware, but this is often feasible because you are rarely using the fastest CPU available.
- Redesign the processes to take less execution time. This requires knowledge of the code and may or may not be possible.
- Rewrite the specification to change the deadlines. This is unlikely to be feasible, but may be in a few cases where some of the deadlines were initially made tighter than necessary.

MULTIPLE TASKS AND MULTIPLE PROCESSES:

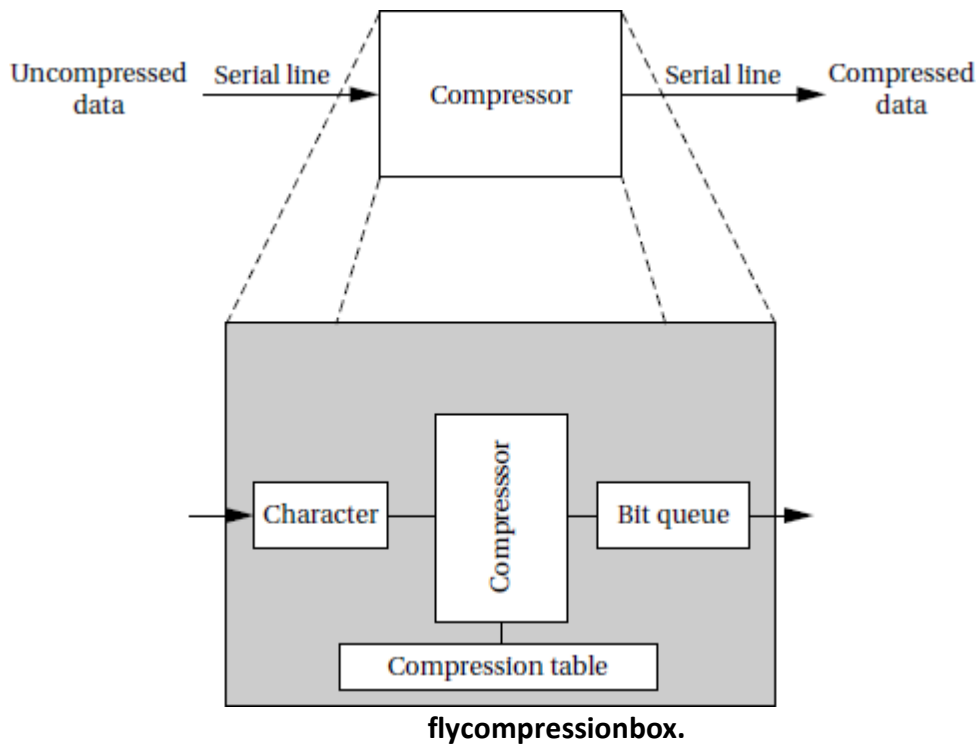
Write short notes on multiple tasks and multiple processes (May 2008)

5.1.1 Tasks and Processes

- ❖ Most of the embedded systems are too complex. Hence programming the systems are also complex. To reduce the complexity we break the system into multiple tasks.
- ❖ Task is nothing but different parts of functionality in a single system. Thus the various application in a system and each system is called Task.
- ❖ For example, when designing a telephone answering machine, we can define recording a phone call and operating the user's control panel as distinct tasks.
- ❖ *A process* is a single execution of a program.
- ❖ If we run the same program two different times, we have created two different processes.
- ❖ Each process has its own state that includes not only its registers but all of its memory.
- ❖ In some OSs, the memory management unit is used to keep each process in a separate address space.
- ❖ In others, particularly lightweight RTOSs, the processes run in the same address space. Processes that share the same address space are often called *threads*.

Example:

- ✓ Consider a standalone compression unit, this device is connected to serial ports on both ends.
- ✓ The input to the box is an uncompressed stream of bytes.
- ✓ The box emits a compressed string of bits on the output serial line, based on a predefined compression table.
- ✓ Such a box may be used, to compress data being sent to a modem.
- ✓ The program's need to receive and send data at different rates.
- ✓ For example, the program may emit 2 bits for the first byte and then 7 bits for the second byte will obviously find itself reflected in the structure of the code.
- ✓ It is easy to create irregular code to solve this problem; a more elegant solution is to create a queue of output bits, with those bits being removed from the queue and sent to the serial port in 8-bit sets.



Anon-the-

- ✓ Ensure that processing of the inputs and outputs are at the proper rates.

- ✓ For example, if too much time is spent in packaging and emitting output characters, we may drop an input character. Solving timing problems is a more challenging problem.
- ✓ The text compression box provides a simple example of rate control problems.
- ✓ A control panel on a machine provides an example of a different type of rate control problem, the *asynchronous input*.
- ✓ The control panel of the compression box may include a compression mode button that disables or enables compression, so that the input text is passed through unchanged when compression is disabled.

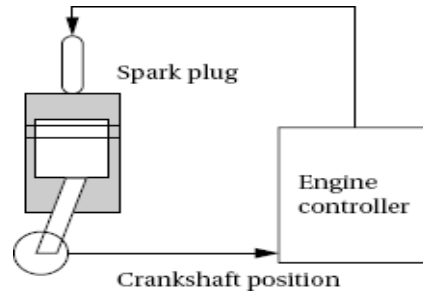
5.1.2 Multirate Systems

Briefly explain about multirate systems with an example. (May 2011)

- The systems which are embedded with more than one application are called multirate systems.
- Implementing code that satisfies timing requirements is even more complex when multiple rates of computation must be handled.
- Multirate embedded computing systems are very common, including automobile engines, printers, and cell phones.
- In all these systems, certain operations must be executed periodically, and each operation is executed at its own rate.

Example: Automotive engine controller:

- ✓ The simplest automotive engine controllers, such as the ignition controller for a basic motorcycle engine, perform only one task, timing the firing of the spark plug, which takes the place of a mechanical distributor.
- ✓ The spark plug must be fired at a certain point in the combustion cycle, but to obtain better performance, the phase relationship between the piston's movement and the spark should change as a function of engine speed.
- ✓ Using a microcontroller that senses the engine crankshaft position allows the spark timing to vary with engine speed.
- ✓ Firing the spark plug is a periodic process.
- ✓ The control algorithm for a modern automobile engine is much more complex, making the need for microprocessors that much greater.
- ✓ Automobile engines must meet strict requirements on both emissions and fuel economy.
- ✓ On the other hand, the engines must still satisfy customers not only in terms of performance, ease of starting in extreme cold and heat and low maintenance.
- ✓ Automobile engine controllers use additional sensors, including the gas pedal position and an oxygen sensor used to control emissions.
- ✓ They also use a multimode control scheme. For example, one mode may be used for engine warm-up, another for cruise, and yet another for climbing steep hills, and so forth.
- ✓ The larger number of sensors and modes increases the number of discrete tasks that must be performed.
- ✓ The engine controller takes a variety of inputs that determine the state of the engine.
- ✓ It controls two basic engine parameters: the spark plug firing and the fuel/air mixture.



5.1.3 Timing Requirements on Processes

Explain in detail about timing requirements on processes.

- ❖ Processes can have several different types of timing requirements imposed on them by the application.
- ❖ The timing requirements on a set of processes strongly influence the type of scheduling that is appropriate.
- ❖ A scheduling policy must define the timing requirements that it uses to determine whether a schedule is valid.

There are two important timing requirements on processes: **release time** and **deadline**.

Release time:

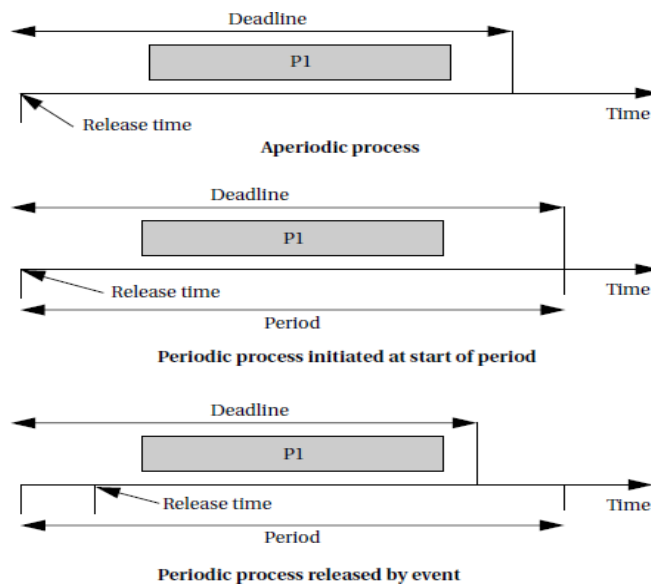
- ✓ The release time is the time at which the process becomes ready to execute.
- ✓ An aperiodic process is initiated by an event, such as external data arriving or data computed by another process.
- ✓ The release time is generally measured from that event. For a periodically executed process, there are two common possibilities.

odically executed process, there are two common possibilities.

- ✓ In simpler systems, the process may become ready at the beginning of the period.
- ✓ More sophisticated systems may set the release time at the arrival time of certain data, at a time after the start of the period.

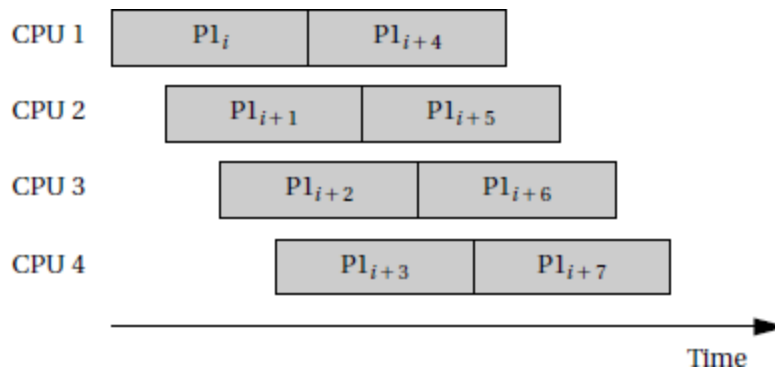
Deadline:

- A deadline specifies when a computation must be finished.
- The deadline for an aperiodic process is generally measured from the release time.
- The deadline for a periodic process may occur at some time other than the end of the period.
- The **period** of a process is the time between successive executions. For example, the period of a digital filter is defined by the time interval between successive input samples.
- The process's **rate** is the inverse of its period. In a multirate system, each process executes at its own distinct rate.



Example definitions of release times and deadlines.

- ✓ For periodic processes the initiation interval to be equal to the period. However, pipelined execution of processes allows the initiation interval to be less than the period.



A sequence of processes with a high initiation rate.

5.1.4 CPU Metrics

Write short notes on CPU Metrics.

CPU metrics are described by **initiation time** and **completion time**

Initiation time:

- The **initiation time** is the time at which a process actually starts executing on the CPU.

Completion time:

- The **completion time** is the time at which the process finishes its work.
- ❖ The most basic measure of work is the amount of **CPU time** expended by a process.
- ❖ The CPU time of process i is called C_i .

- ❖ The CPU time is not equal to the completion time minus initiation time; several other processes may interrupt execution.
- ❖ The total CPU time consumed by a set of processes is

$$T = \sum_{1 \leq i \leq n} T_i.$$

- ❖ To measure the efficiency of CPU, the simplest and most direct measure is *utilization*:

$$U = \frac{\text{CPU time for useful work } T}{\text{total available CPU time } t}$$

$$U = T/t$$

- ❖ Utilization is the ratio of the CPU time that is being used for useful computations to the total available CPU time.
- ❖ This ratio ranges between 0 and 1, with 1 meaning that all of the available CPU time is being used for system purposes.
- ❖ The utilization is often expressed as a percentage.

5.2 OPERATING SYSTEMS:

Discuss in detail about Preemptive real time operating systems. (NOV/DEC 2007, May 2012, NOV 2017, APRIL/MAY 2019)

- ❖ An Operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.
- ❖ A more common definition is that the operating system is the one program running at all times on the computer (usually called the kernel), with all application programs.
- ❖ An Operating system is concerned with the allocation of resources and services, such as memory, processors, devices and information. The Operating System correspondingly includes programs to manage these resources, such as a traffic controller, a scheduler, memory management module, I/O programs, and a file system.

PREEMPTIVE REAL-TIME OPERATING SYSTEMS

- A RTOS executes processes based upon timing constraints provided by the system designer.
- The most reliable way to meet timing constraints accurately is to build a *preemptive OS* and to use *priorities* to control what process runs at any given time. This operating system runs on many different platforms.

Preemption

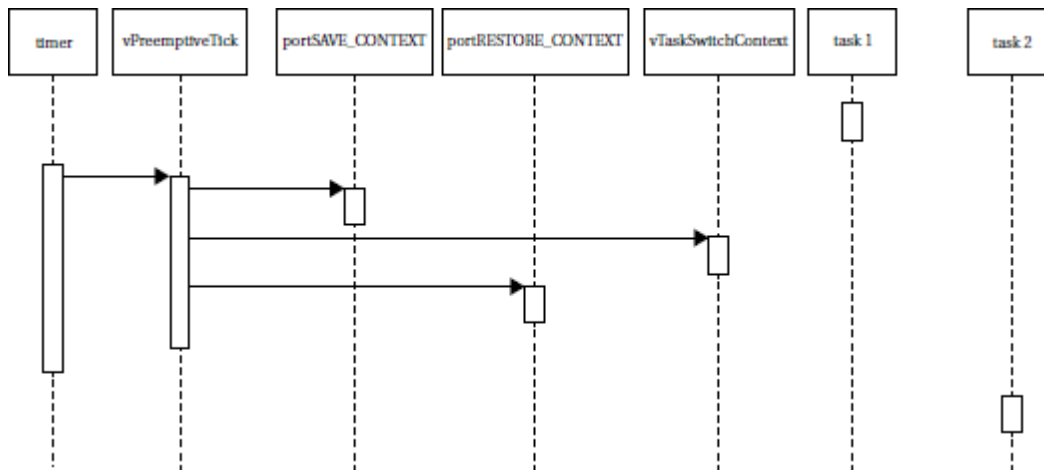
- Preemption is an alternative to the C function call as a way to control execution.
- To be able to take full advantage of the timer, assume a process as something more than a function call.
- Create new routines that allow us to jump from one subroutine to another at any point in the program.

- That, together with the timer, will allow moving between functions whenever necessary based up on the system's timing constraints.
- The CPU is shared across two processes. The *kernel* is the part of the OS that determines what process is running.
- The kernel is activated periodically by the timer.
- The length of the timer period is known as the *time quantum* because it is the smallest increment in which we can control CPU activity.
- The kernel determines what process will run next and causes that process to run.
- On the next timer interrupt, the kernel may pick the same process or another process to run.
- Before, using the timer to control loop iterations, with one loop iteration including the execution of several complete processes.
- Here, the time quantum is in general smaller than the execution time of any of the processes.
- The timer interrupt causes control to change from the currently executing process to the kernel; as assembly language can be used to save and restore registers.
- Similarly use assembly language to restore registers not from the process that was interrupted by the timer but to use registers from any process we want.
- The set of registers that define a process are known as its *context* and switching from one process's registers set to another is known as *context switching*.
- The data structure that holds the state of the process is known as the *process control block*.

PROCESSES AND CONTEXT SWITCHING:

Write short notes on context switching.

- ✓ The first job of the OS is to determine the process that runs next.
- ✓ The work of choosing the order of running processes is known as scheduling.
- ✓ The OS considers a process to be in one of three basic *scheduling states*: *waiting*, *ready*, or *executing*.
- ✓ The best way to understand processes and context is to dive into an RTOS implementation.
- ✓ A process is known in FreeRTOS.org as a task.
- ✓ Task priorities in FreeRTOS.org are ranked opposite to the convention, higher numbers denote higher priorities and the priority 0 task is the idle task.



SequencediagramforfreeRTOS.org contextswitch.

- ✓ To understand the basics of a context switch, assume that the set of tasks is in a steady state.
- ✓ Everything has been initialized, the OS is running, and we are ready for a timer interrupt.
- ✓ These sequencediagram show the application tasks, the hardware timer, and all the functions in the kernel that are involved in the context switch:
 - vPreemptiveTick() is called when the timer ticks.
 - PortSAVE_CONTEXT() swaps out the current task context..
 - vTaskSwitchContext() chooses a new task.
 - PortRESTORE_CONTEXT() swaps in the new context.

5.4 SCHEDULING POLICIES:

Explain in detail about different types of scheduling policies. (April 2013)

A *scheduling policy* defines how processes are selected for promotion from the ready state to the running state.

- ✚ Choosing the right scheduling policy not only ensures that the system will meet all its timing requirements, it also influences the CPU horsepower required to implement the system's functionality.
- ✚ Schedulability means whether there exists a schedule of execution for the processes in a system that satisfies all their timing requirements.
- ✚ Utilization is one of the key metrics in evaluating a scheduling policy. The most basic requirement is that CPU utilization be no more than 100%.
- ✚ For periodic processes, the length of time that must be considered is the *hyperperiod*, which is the least-common multiple of the periods of all the processes.
- ✚ The complete schedule for the least-common multiple of the periods is sometimes called the *unrolled schedule*.

Types of scheduling:

Cyclostatic Scheduling:

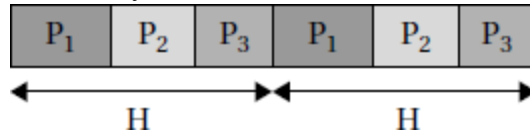
One very simple scheduling policy is known as *cyclostatic* scheduling or sometimes as

Time Division Multiple Access scheduling.

A cyclostatic schedule is divided into equal-

sized time slots over an interval equal to the length of the hyperperiod H .

Processes always run in the same time slot.



Cyclostatic scheduling.

Two factors affect utilization in cyclostatic scheduling

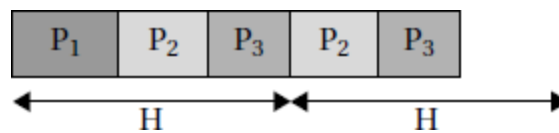
- The number of time slots used
- The fraction of each time slot that is used for useful work.

Depending on the deadlines for some of the processes, sometimes slots may need to be left empty.

Since the time slots are of equal size, some short processes may have time left over in their time slot

Round-robin scheduling:

- ❖ Round-robin uses the same hyperperiod as in cyclostatic.
- ❖ It also evaluates the processes in order.



Round-robin scheduling.

- ❖ But unlike cyclostatic scheduling, if a process does not have any useful work to do, the round-robin scheduler moves on to the next process in order to fill the time slot with useful work.
- ❖ In this example, all three processes execute during the first hyperperiod, but during the second one, P_1 has no useful work and is skipped.
- ❖ The processes are always evaluated in the same order.
- ❖ The last time slot in the hyperperiod is left empty; if we have occasional, non-periodic tasks without deadlines.
- ❖ Round-robin scheduling is often used in hardware such as buses because it is very simple to implement but it provides some amount of flexibility.

- ❖ In addition to utilization, also considers **scheduling overhead**—the execution time required to choose the next execution process, which is incurred in addition to any context switching overhead.
- ❖ In general, the more sophisticated the scheduling policy, the more CPU time it takes during system operation to implement it
- ❖ The final decision on a scheduling policy must take into account both theoretical utilization and practical scheduling overhead.

5.5 PRIORITY-BASED SCHEDULING

Briefly explain about priority based scheduling and its types (NOV/DEC 2006, 2007, 2009, May 2012, May 2023, Dec 2023)

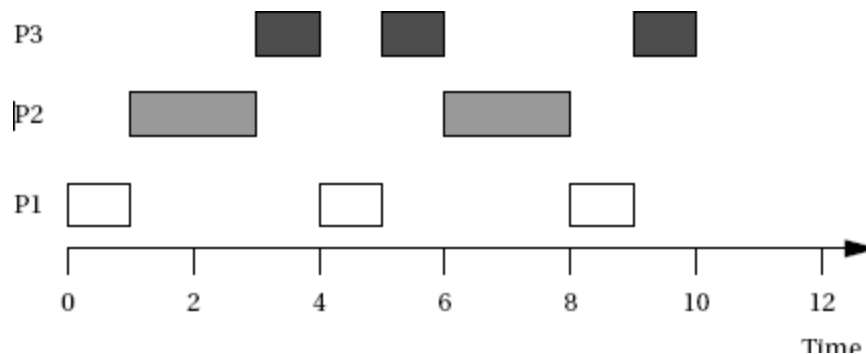
- ❖ To determine an algorithm to assign priorities to the processes, the OS takes care of them by choosing the highest-priority ready process. There are two major ways to assign priorities:
 - I) Static: Static* priorities are that do not change during execution
 - II) Dynamic: Dynamic* priorities that do change during execution.
 Depending on the static and dynamic way of assigning priority there are two methods to schedule the process. They are

Rate-Monotonic Scheduling

- ❖ **Rate-monotonic scheduling (RMS)**, introduced by Liu and Layland [Liu73], was one of the first scheduling policies developed for real-time systems and is still very widely used.
- ❖ RMS is a static scheduling policy.
- ❖ It turns out that these fixed priorities are sufficient to efficiently schedule the processes in many situations.
- ❖ The theory underlying RMS is known as **rate-monotonic analysis (RMA)**. This theory, as summarized below.
 - ✓ All processes run periodically on a single CPU.
 - ✓ Context switching time is ignored. There are no data dependencies between processes.
 - ✓ The execution time for a process is constant.
 - ✓ All deadlines are at the ends of their periods.
 - ✓ The highest-priority ready process is always selected for execution.
- ❖ The major result of RMA is that a relatively simple scheduling policy is optimal under certain conditions.
- ❖ Priorities are assigned by rank order of period, with the process with the shortest period being assigned the highest priority.
- ❖ This fixed-priority scheduling policy is the optimum assignment of static priorities to processes, in that it provides the highest CPU utilization while ensuring that all processes meet their deadlines.
- ❖ Here is a simple set of processes and their characteristics.

Process	Execution time	Period
P1	1	4
P2	2	6
P3	3	12

- ❖ Applying the principles of RMA, we give P1 the highest priority, P2 the middle priority, and P3 the lowest priority.
- ❖ To understand all the interactions between the periods, construct a timeline equal in length to hyperperiod, which is 12 in this case.



- ❖ All three periods start at time zero. P1's data arrive first. Since P1 is the highest-priority process, it can start to execute immediately.
- ❖ After one time unit, P1 finishes and goes out of the ready state until the start of its next period. At time 1, P2 starts executing as the highest-priority ready process.
- ❖ At time 3, P2 finishes and P3 starts executing. P1's next iteration starts at time 4, at which point it interrupts P3.
- ❖ P3 gets one more time unit of execution between the second iterations of P1 and P2, but P3 does not get to finish until after the third iteration of P1.
- ❖ In this case the CPU time executed within the period of 12 units. Even though each process alone has an execution time significantly less than its period, combinations of processes can require more than 100% of the available CPU cycles.

Response Time: Response time of a process is the time at which the process finishes.

Critical Instant: The *critical* instant for a process is defined

as the instant during execution at which the task has the largest response time.

- ❖ The proof using critical instants is easy while knowing the RMA when it is ready and all higher priority processes are also ready.
- ❖ Also critical instant is used to determine whether there is any feasible schedule for the system.
- ❖ Critical-instant analysis also implies that priorities should be assigned in order of periods.

Earliest–Deadline–First scheduling:

Write short notes on Earliest–Deadline– First scheduling.

- ❖ *Earliest deadline first (EDF)* is another well-known scheduling policy
- ❖ It is a dynamic priority scheme, it changes process priorities during execution based on initiation times.
- ❖ As a result, it can achieve higher CPU utilization than RMS.
- ❖ The EDF policy is also very simple.
- ❖ It assigns priorities in order of deadline.
- ❖ The highest-priority process is the one whose deadline is nearest in time, and the lowest priority process is the one whose deadline is farthest away.
- ❖ Clearly, priorities must be recalculated at every completion of a process.
- ❖ The final step of the OS during the scheduling procedure is the same as for RMS and the highest-priority ready process is chosen for execution.

Example:

For Earliest–deadline–first scheduling

Consider the following processes:

Process	Execution time	Period
P1	1	3
P2	1	4
P3	2	5

The hyperperiod is 30. According to the above system, P1 has the highest priority, P2 is the middle priority, and P3 is the lowest priority. Then the deadline table is written as

Time	Running process	Deadline
0	P1	
1	P2	
2	P3	
3	P3	
4	P1	P1
5	P2	P2
6	P1	P3
7	P3	P1
8	P3	
9	P1	P2
10	P2	P1
11	P3	P3
12	P1	
13	P3	P1,P2
14	P2	P2
15	P1	P2,P3
16	P2	
17	P3	P1
18	P1	P2,P3
19	Idle	
20	P3	P1
21	P2	
22	P1	P2,P3
23	P3	P1
24	P3	P2
25	P1	P3
26	P2	P1,P2
27	P2	P3
28	P1	P1
29	P3	P2,P3

The one time slot is idle at $t = 19$ then the CPU utilization is $19/30$. Hence the EDF is achieved nearly to 100% utilization of CPU.

Compare RMS versus EDF (NOV/DEC 2018) RMS vs EDF

Compare Rate-Monotonic Scheduling and Earliest-Deadline-First scheduling

Which scheduling policy is better: RMS or EDF? That depends on the criteria.

- ❖ EDF can extract higher utilization out of the CPU, but it may be difficult to diagnose the possibility of an imminent overload.
- ❖ RMS achieves lower CPU utilization but it is easier to ensure that all deadlines. Able to diagnose the possibility of an imminent overload.

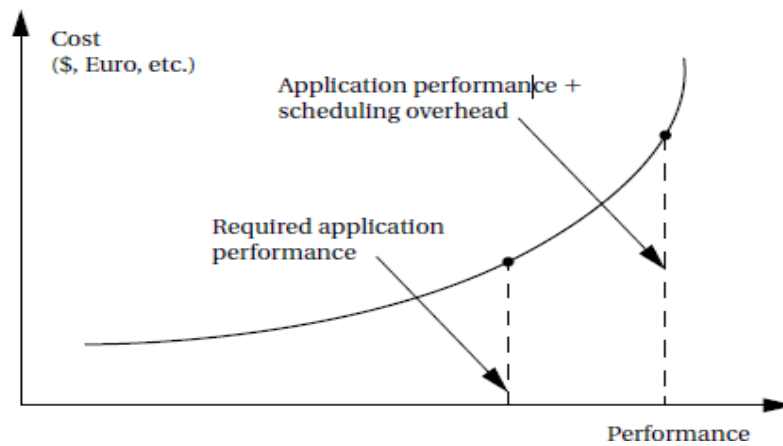
If a set of processes is unschedulable and we need to guarantee that they complete their deadlines? There are several possible ways to solve this problem:

- **Get a faster CPU.** That will reduce execution times without changing the periods, giving you lower utilization.
- **Redesign the processes to take less execution time.** This requires knowledge of the code and may or may not be possible.
- **Rewrite the specification to change the deadlines.** This is unlikely to be feasible, but may be in a few cases where some of the deadlines were initially made tighter than necessary.

5.6. MULTIPROCESSOR:

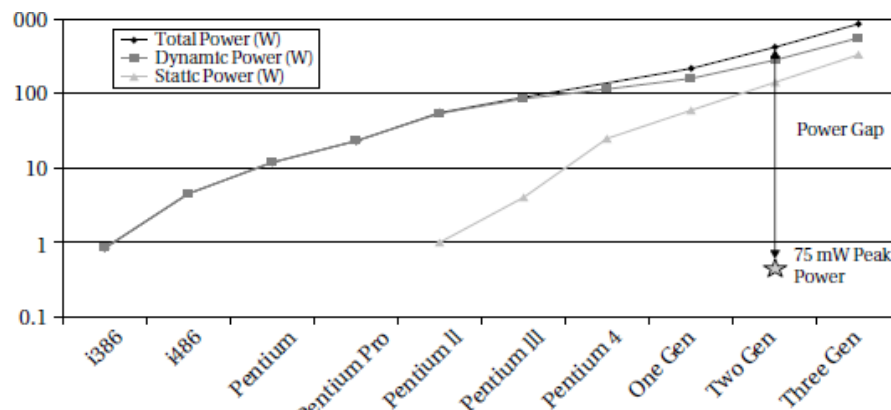
Write short notes on Multiprocessor (April 2010)

- A **multiprocessor** is, in general, any computer system with two or more processors coupled together.
- Multiprocessors used for scientific or business applications tend to have regular architectures: several identical processors that can access a uniform memory space.
- Embedded system designers must take a more general view of the nature of multiprocessors.
- The first reason for using an embedded multiprocessor is that they offer significantly better cost/performance that is, performance and functionality per dollar spent on the system.
- The cost of a microprocessor increases greatly as the clock speed increases.
- Clock speeds are normally distributed by normal variations in VLSI processes; because the fastest chips are rare, they naturally command a high price in the marketplace.
- Because the fastest processors are very costly, splitting the applications so that it can be performed on several smaller processors is usually much cheaper.
- Even with the added cost of assembling those components, the total system comes out to be less expensive.
- In addition to reducing costs, using multiple processors can also help with real-time performance.
- It may take an extremely large and powerful CPU to provide the same responsiveness that can be had from a distributed system.
- Many of the technology trends encourage us to use multiprocessors for performance also lead us to multiprocessing for low power embedded computing.
- Some processors running at slower clock rates consume less power than a single large processor: performance scales linearly with power supply voltage but power scales with V^2 .



Scheduling overhead is paid for at a nonlinear rate

- Austin *et al.* [Aus04] shows that general-purpose computing platforms are not keeping up with the strict energy budgets of battery-powered embedded computing.



Power consumption trends for desktop processors [Aus04].

- Desktop processors require close to 1000 times that amount of power to run.
- That huge gap cannot be solved by weakening processor architectures or software.
- Multiprocessors provide a way to break this power barrier and build substantially more efficient embedded computing platforms.

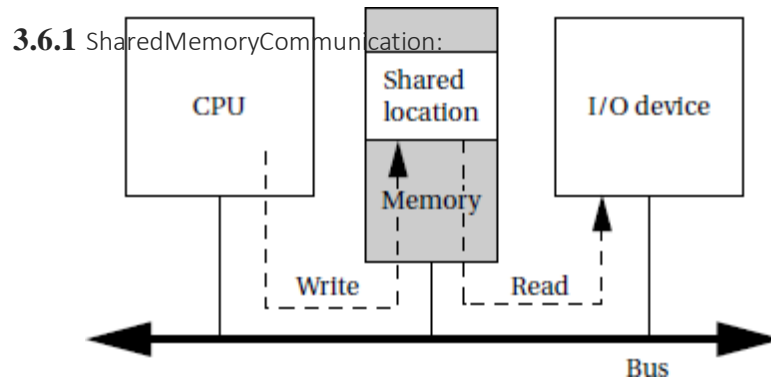
5.7. INTERPROCESS COMMUNICATION MECHANISMS: Explain in detail about interprocess communication mechanisms.

(NOV/DEC 2010, May 2014, NOV 2017, April 2018, NOV/DEC 2018, APRIL/MAY

2019, APRIL/MAY 2023)

- ✓ Processes often need to communicate with each other. *Interprocess communication mechanisms* are provided by the operating system as part of the process abstraction.
- ✓ The process can send a communication in one of two ways: **blocking** or **non-blocking**. **Blocking communication**: The process goes into the waiting state until it receives a response called blocking communication.
- ✓ **Non-blocking communication**: It allows the process to continue execution after sending the communication. Both types of communication are useful.

There are two major styles of interprocess communication: **shared memory** and **message passing**.



Shared memory communication implemented on a bus.

- Shared memory communication works in a bus-based system.
- Two components, such as a CPU and an I/O device, communicate through a shared memory location.
- The software on the CPU has been designed to know the address of the shared location.
- The shared location has also been loaded into the proper register of the I/O device.
- If the CPU wants to send data to the device, it writes to the shared location.
- The I/O device then reads the data from that location. The read and write operations are standard and can be encapsulated in a procedural interface.
- If the CPU and the I/O device want to communicate through a shared memory block, there must be a flag that tells the CPU when the data from the I/O device is ready.
- The flag, an additional shared data location, has a value of 0 when the data are not ready and 1 when the data are ready.
- If the flag is used only by the CPU, then the flag can be implemented using a standard memory write operation.
- If the same flag is used for bidirectional signaling between the CPU and the I/O device, care must be taken.

To care the flag following scenario must be followed:

1. CPU reads the flag location and sees that it is 0.
2. I/O device reads the flag location and sees that it is 0.
3. CPU sets the flag location to 1 and writes data to the shared location.
4. I/O device erroneously sets the flag to 1 and overwrites the data left by the CPU.

By using the bidirectional flag a critical timing race between the two programs is caused.

To avoid this, the microprocessor bus must support an atomic test-and-set operation.

Test-and-Set operation:

- The test-and-set operation first reads a location and then sets it to a specified value. It returns the result of the test.
- If the location was already set, then the additional set has no effect but the test-and-set instruction returns a false result.
- If the location was not set, the instruction returns true and the location is in fact set. The bus supports this as an *atomic* operation that cannot be interrupted. A test-and-set can be used to implement a *semaphore*.

Semaphore:

- ✓ *Semaphore* is a language-level synchronization construct. Let's assume that the system provides one semaphore that is used to guard access to a block of protected memory.
- ✓ Any process that wants to access the memory must use the semaphore to ensure that no other process is actively using it.

Test-and-set operation example:

✓ The SWP (swap) instruction is used in the ARM to implement atomic test-and-set: SWP Rd, Rm, Rn

- ✓ The SWP instruction takes three operands—the memory location pointed to by *Rn* is loaded and saved into *Rd*, and the value of *Rm* is then written into the location pointed to by *Rn*.
- ✓ When *Rd* and *Rn* are the same register, the instruction swaps the register's value and the value stored at the address pointed to by *Rd/Rn*. For example, consider this code sequence:

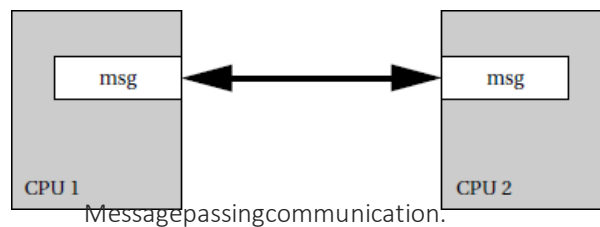
```
ADR r0, SEMAPHORE: get semaphore address
DR r1, #1
GETFLAG SWP r1, r1, [r0]: test-and-set the
flag BNZ GETFLAG; no flag yet, try again
HASFLAG
```

- ✓ The program first loads the constant 1 into *r1* and the address of the semaphore FLAG1 into register *r2*, then reads the semaphore into *r0* and writes the 1 value into the semaphore.
- ✓ The code then tests whether the semaphore fetched from memory is zero; if it was, the semaphore was not busy and we can enter the critical region that begins with the HASFLAG label.

- ✓ If the flag was non-zero, we loop back to try to get the flag once again.

5.7.1. Message Passing:

- ❖ Message passing communication complements the shared memory model: each communicating entity has its own message send/receive unit.
- ❖ The message is not stored on the communication link, but rather at the senders/receivers at the endpoints.
- ❖ In contrast, shared memory communication can be seen as a memory block used as a communication device, in which all the data are stored in the communication link/memory.

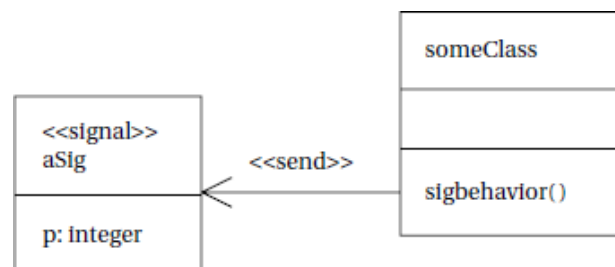


- ❖ Applications in which units operate relatively autonomously are natural candidates for message passing communication.
- ❖ For example, a home control system has one microcontroller per household device—lamp, thermostat, faucet, appliance, and so on.
- ❖ The devices must communicate relatively infrequently; furthermore, their physical separation is large enough that we would not naturally think of them as sharing a central pool of memory.
- ❖ Passing communication packets among the devices is a natural way to describe coordination between these devices.
- ❖ Message passing is the natural implementation of communication in many 8-bit microcontrollers that do not normally operate with external memory.

5.7.2. Signals

Another form of interprocess communication commonly used in UNIX is the **signal**.

- ✚ A signal is simple because it does not pass data beyond the existence of the signal itself. A signal is analogous to an interrupt, but it is entirely a software creation.
- ✚ A signal is generated by a process and transmitted to another process by the operating system.



Use of a UML signal.

- ✦ A UML signal is actually a generalization of the UNIX signal.
- ✦ While a UNIX signal carries no parameters other than a condition code, a UML signal is an object.
- ✦ The *sigbehavior()* behavior of the class is responsible for throwing the signal, as indicated by `<<send>>`. The signal object is indicated by the `<<signal>>` stereotype.

to save and restore context and we must execute additional instructions to implement the scheduling policy.

- ❖ On the other hand, context switching can be implemented efficiently; context switching need not kill performance.
- ❖ The effects of nonzero context switching time must be carefully analyzed in the context of a particular implementation to be sure that the predictions of an ideal scheduling policy are sufficiently accurate.
- ❖ In most real-time operating systems, a context switch requires only a few hundred instructions, with only slightly more overhead for a simple real-time scheduler like RMS.
- ❖ When the overhead time is very small relative to the task periods, then the zero-time context switch assumption is often a reasonable approximation.
- ❖ Problems are most likely to manifest themselves in the highest-rate processes, which are often the most critical in any case.
- ❖ Completely checking that all deadlines will be met with nonzero context switching time requires checking all possible schedules for processes and including the context switch time at each preemption or process initiation.

5.8.3. DISTRIBUTED EMBEDDED SYSTEMS

1. Discuss in detail about the distributed embedded architecture. (Nov/Dec 2014, April 2018, NOV/DEC 2018)

2. Explain about distributed embedded architecture with suitable examples.

(Apr/May 2015)(Dec 2022/Jan 2023)

- Distributed system is more than two CPUs communicated in a tightly coupled manner.
- The main reason to implementing distributed system in the embedded field is it will provide high performance to perform complicated task in an easy manner, high processing rate.

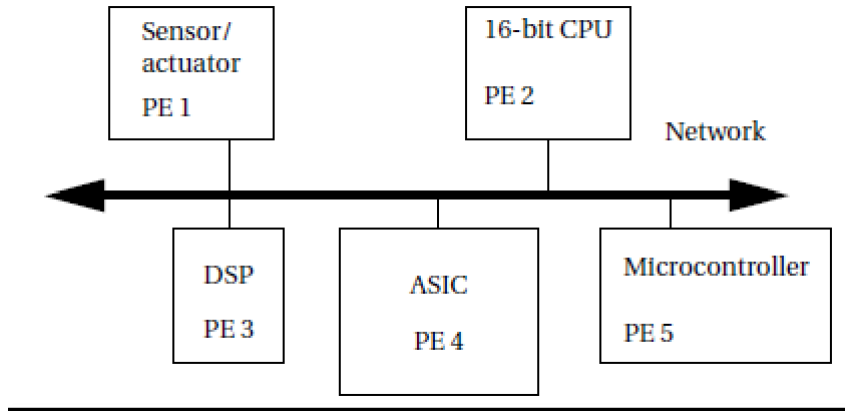
Types

1. System Architecture

2. Software Architecture

- A distributed embedded system can be organized in many different ways, but its basic units are PE and the network processing element (PE) is either microprocessors or ASIC used to connect by a network.
- Processing element allows the network to communicate.
- A distributed embedded system can be organized in many different ways,
- A PE may be an instruction set processor such as a *DSP*, *CPU*, or *microcontroller*, as well as a non-programmable unit such as the *ASICs* used to implement *PEs*.
- An I/O device such as *PE I* (which we call here a *sensor* or *actuator*, depending on whether it provides input or output) may also be a PE, so long as it can speak the network protocol to communicate with other PEs.
- The network in this case is a bus, but other network topologies are also possible.
- It is also possible that the system can use more than one network, such as when relatively independent functions require relatively little communication among them.
- We often refer to the connection between PEs provided by the network as a **communication link**.
- The system of PEs and networks forms the **hardware platform** on which the application runs.
- In particular, PEs do not fetch instructions over the network as they do on the microprocessor bus

- The speed at which PE can communicate over the bus would be difficult if not impossible to predict if we allowed arbitrary instruction and data fetches as we do on microprocessor buses.



An example of Distributed Embedded System

5.8.4. REASON FOR DISTRIBUTED SYSTEM IN EMBEDDED SYSTEM

In distributed embedded system having several PEs and network it leads to more complicated than using a single large microprocessor to perform the same tasks.

1. In distributed system the PE communicates with physically separated manner.
2. Deadlines for processing the data are short.
3. It has more cost-effective performance
4. One part of the system can be used to help solve problems in another part.
5. Error identification is easier.
6. Several CPUs involved in the processing
7. One CPU is used to generate inputs for another CPU and watch its output.

4.5.2. NETWORK ABSTRACTIONS

- Networks are complex systems. Ideally, they provide high-level services while hiding many of the details of data transmission from the other components.
- In order to help understand (and design) networks, the International Standards Organization has developed a seven-layer model for networks known as Open Systems Interconnection (OSI) models.
- Understanding the OSI layers will help us to understand the details of real networks.
- These seven layers of the **OSI model**, are intended to cover a broad spectrum of networks and their uses.

- Some networks may not need the services of one or more layers because the higher layers may be totally missing or an intermediate layer may not be necessary.
- However, any data network should fit into the OSI model.

Application	End-use interface
Presentation	Data format
Session	Application dialog control
Transport	Connections
Network	End-to-end service
Data link	Reliable data transport
Physical	Mechanical, electrical

The OSI layers from lowest to highest level of abstraction are described below.

- ✓ Physical Layer
 - The physical layer defines the basic properties of the interface between systems, including the physical connections, electrical properties, basic functions of the electrical and physical components and the basic procedures for exchanging bits.
- ✓ Data Link Layer
 - The primary purpose of this layer is error detection and control across a single link.
 - If the network requires multiple hops over several data links, the data link layer does not define the mechanism for data integrity between hops, but only within a single hop.
- ✓ Network Layer
 - This layer defines the basic end-to-end data transmission service. The network layer is particularly important in multi-hop networks.
 - This layer divides the message into packet form.
- ✓ Transport Layer
 - The transport layer defines connection oriented services that ensures that data are delivered in the proper order and without errors across multiple links.
 - This layer may also try to optimize network resource utilization.
- ✓ Presentation Layer:
 - Presentation layer defines data exchange formats and provides transformation utilities to application programs.
- ✓ Session Layer
 - A session provides mechanisms for controlling the interaction of end user services across a network, such as data grouping and checkpointing.
- ✓ Application Layer
 - The application layer provides the application interface between the network and end-user programs.

- Simple embedded networks provide internet service that will implement the full range of functions in the OSI model.

NETWORKS FOR EMBEDDED SYSTEMS

- Networks for embedded computing span a broad range of requirements; many of those requirements are very different from those for general-purpose networks.
- Some networks are used in safety-critical applications, such as automotive control.
- Some networks, such as those used in consumer electronics systems, must be very inexpensive.
- Other networks, such as industrial control networks, must be extremely rugged and reliable.

Several interconnect networks have been developed especially for distributed embedded computing:

- ✓ The I²C bus is used in microcontroller-based systems.
- ✓ The Controller Area Network (CAN) bus was developed for automotive electronics. It provides megabit rates and can handle large numbers of devices.
- ✓ Ethernet and variations of standard Ethernet are used for a variety of control applications

MPSOCs AND SHARED MEMORY MULTIPROCESSORS

5.9.5. Write short notes on MPSoCs and Shared memory multiprocessors (NOV 2017) (Dec 2022/Jan 2023)

- Shared memory processors are well-suited to applications that require a large amount of data to be processed. Signal processing systems stream data and can be well-suited to shared memory processing.
- Most MPSoCs are shared memory systems. Shared memory allows for processors to communicate with varying patterns. If the pattern of communication is very fixed and if the processing of different steps is performed in different units, then a networked multiprocessor may be most appropriate.
- If the communication patterns between steps can vary, then shared memory provides that flexibility. If one processing element is used for several different steps, then shared memory also allows the required flexibility in communication.

Heterogeneous shared memory multiprocessors

- Many high-performance embedded platforms are heterogeneous multiprocessors.
- Different processing elements perform different functions. The PEs may be programmable processors with different instruction sets or specialized accelerators that provide little or no programmability.

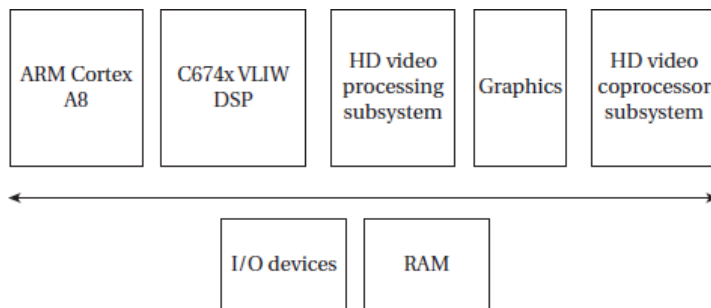
- In both cases, the motivation for using different types of PEs is efficiency. Processors with different instruction sets can perform different tasks faster and using less energy. Accelerators provide even faster and lower-power operation for a narrow range of functions.

Example: TITMS320DM816x DaVinci

- The Da Vinci 816x is designed for high-performance video applications.
- It includes both a CPU, a DSP, and several specialized units: The 816x has two main programmable processors.
- The ARM Cortex A8 includes the Neon multimedia instructions. It is an in-order dual-issue machine.

The C674x is a VLIW DSP. It has six ALUs and 64 general-purpose registers.

- The HD video coprocessor subsystem (HDVICP2) provides image and video acceleration.
- It natively supports several standards, such as H.264 (used in BluRay), MPEG-4, MPEG-2 and JPEG.



- It includes specialized hardware for major image and video operations, including transform and quantization, motion estimation, and entropy coding. It also has its own DMA Engine.
- It can operate at resolutions up to 1080P/60 frames/sec. The HD video processing subsystem (HDVPSS) provides additional video processing capabilities. It can process up to three high-definition and one standard-definition video streams simultaneously.
- It can perform operations such as scan rate conversion, Chroma key, and video security. The graphics unit is designed for 3D graphics operation that can process up to 30M triangles/sec.

AUDIOPLAYER

5.9a.1. Design an Audio Player (NOV 2017)

- ❖ Audio players are defined as any media player which can only play audio files.
- ❖ Players capable of video playback are included under comparison of video player software, even if they are primarily well known for audio playback.

Theory of operation and requirements:

- ❖ Audio players are often called MP3 players.
- ❖ After the popular audio data format, a number of audio compression formats have been developed and are in regular use.
- ❖ The earliest portable MP3 players were based on compact disc mechanisms and modern MP3 players use either flash memory or disk drives to store music.
- ❖ Functions:
 - An MP3 player performs three basic functions such as
 1. Audio storage
 2. Audio compression
 3. User Interface
- ❖ **Audio decompression:** It is relatively light weight. The incoming bit stream has been encoded using a Huffman style code, which must be decoded. The audio data itself is applied to reconstruction filter, along with a number of other parameters.
- ❖ **Audio compression:** It is a lossy process that relies on perceptual coding. The coder eliminates certain features of the audio stream so that the result can be encoded in fewer bits. It tries to eliminate features that are not easily perceived by human audio system.

- ❖ **Masking:** It is one perceptual phenomenon that is exploited by perceptual coding. One tone can be masked by another if the tones are sufficiently close in frequency. Some audio features can also be masked if they occur too close in time after another feature.

MPEG Layer 1 encoder

Filterbank: It splits the signal into a set of 32 subbands that are equally spaced in the frequency domain and together cover the entire frequency range of the audio.

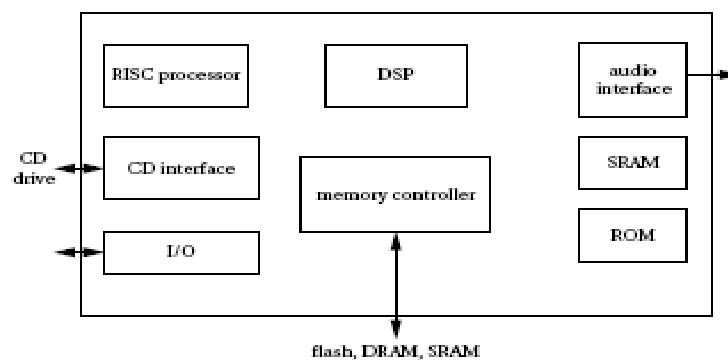


FIGURE 7.22
Architecture of a Cirrus audio processor for CD/MPEG players.

- ❖ **Encoder:** Audio signals tend to be more correlated within a narrower band, so splitting into subbands helps the encoder reduce the bitrate.
- ❖ **Quantizer:** It scales each sub band so that it fits within 6 bits of dynamic range, then quantizes based upon the current scale factor for that subband.
- ❖ **Masking model:** It selects the scale factors. It is driven by a separate Fast Fourier Transform (FFT), the filter bank could be used for masking; a separate FFT provides better results.
- ❖ **Multiplexer:** The multiplexer at the output of the encoder passes along all the required data.

MPEG Layer 1 decoder:

- MPEG audio decoding is a straightforward process.
- After disassembling the data frame, the data are unscaled and inverse quantized to produce sample streams for the sub band.
- An inverse filter bank reassembles the subbands into the uncompressed data.

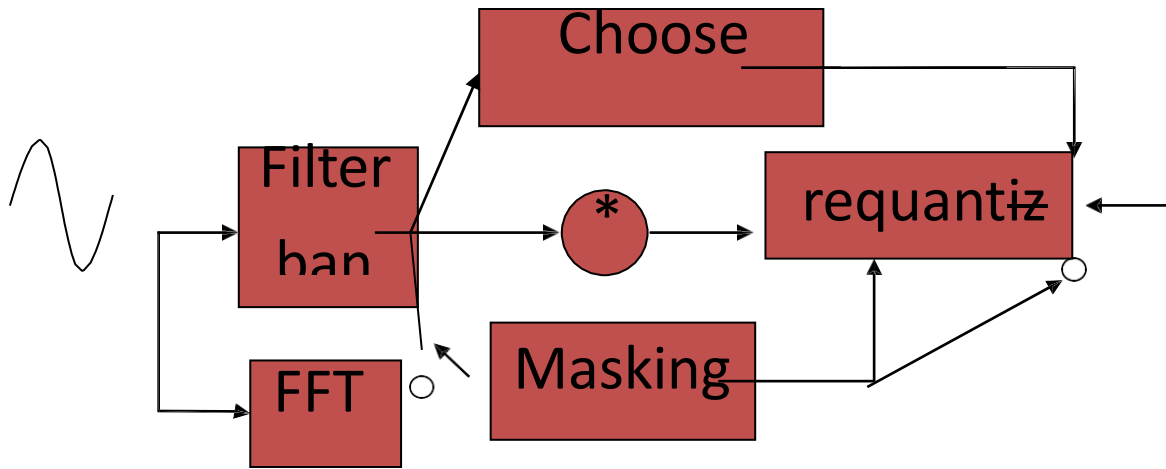


Fig.MPEGLayer1encoder

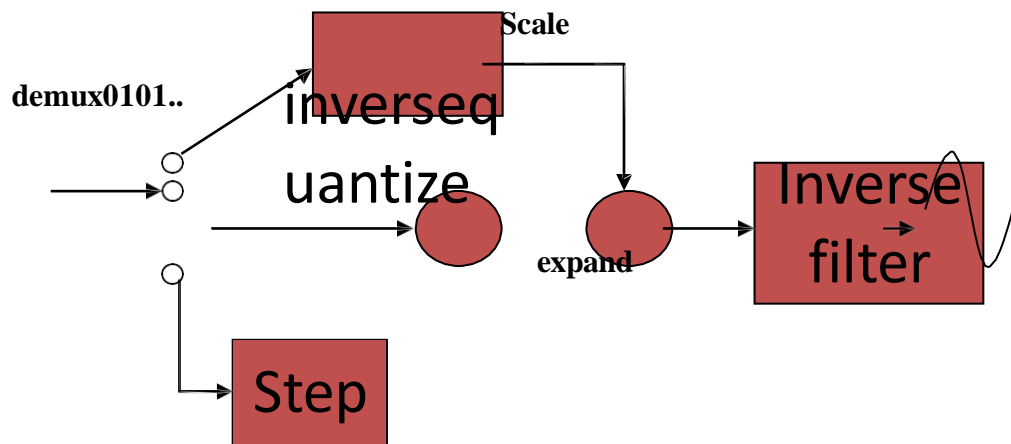


Fig.MPEGLayer1decoder

User interface: The user interface of an MP3 player is usually kept simple to minimize both physical size and power consumption of the device. Many players provide only a simple display and a few buttons.

File system: The file system of the player must be **compatible** with PCs. The CD/MP3 players used compact discs that had been created on PCs. Today's players can be plugged into USB ports and treated as disk drivers on the host processor.

Specification:

- ✓ The file ID class is an abstraction of a file in the flash file system. The controller class provides the method that operates the player.

✓ The file management is performed on a host device, then the basic operations to be specified are simple.

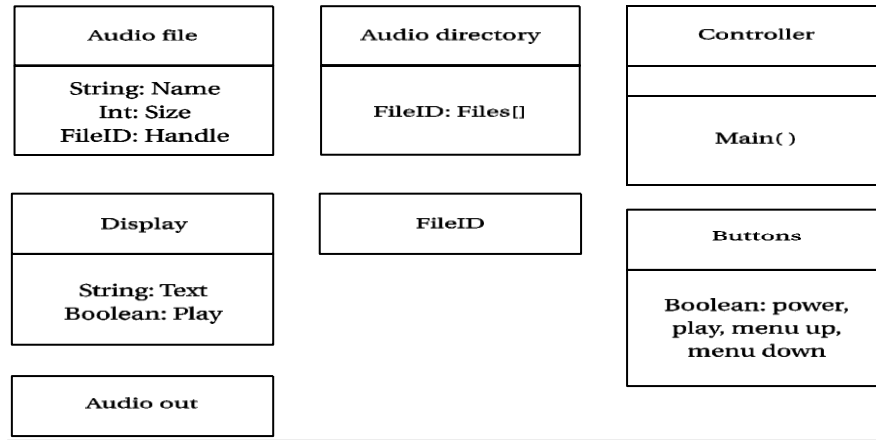


Fig. Classes in the audioplayer state diagram

am for file display/Selection:

This specification assumes that all files are in the root directory and the files are playable audio.

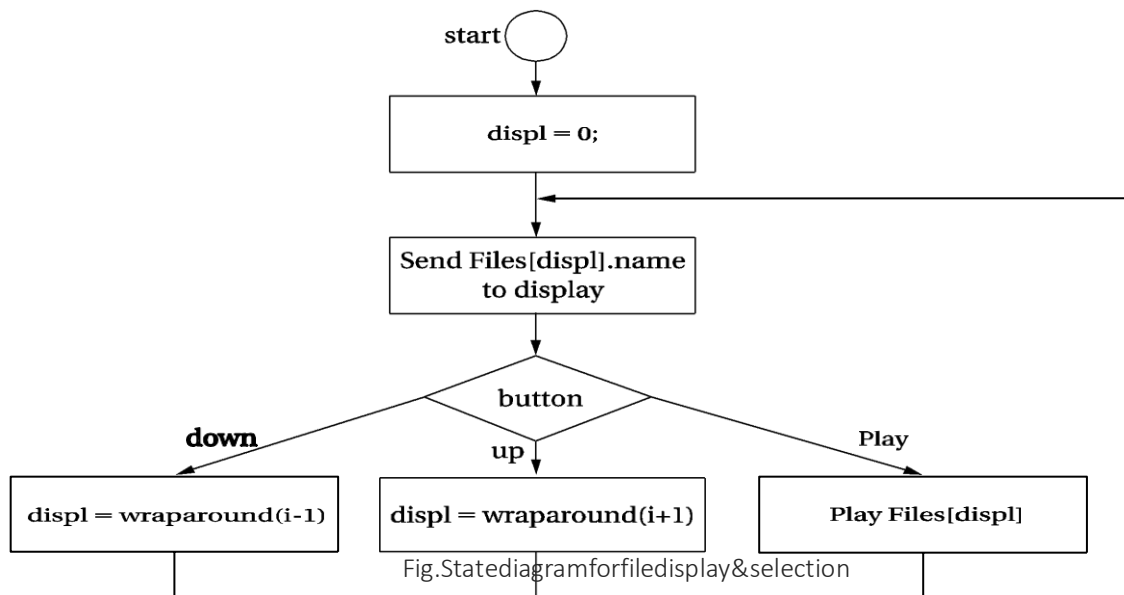
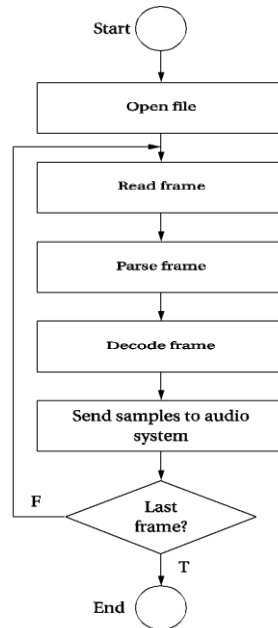


Fig. State diagram for file display & selection

State diagram for audio playback:

✓ The details of this operation depend on the format of the audio file.

- ✓ This state diagram refers to send the samples to the audio system rather than explicitly sending them because playback and reading the next data frame must be overlapped to ensure continuous operation.



State diagram for audio playback

System Architecture:

The cirrus CS7410 is an audio controller designed for CD/MP3 players. The audio controller includes two processors.

1. The 32bit RISC processor is used to perform system control and audio decoding.
2. The 16bit DSP is used to perform audio effects such as equalization.

The memory controller can be interfaced to several different types of memory such as

1. Flash
2. DRAM
3. SRAM

- ❖ Flash memory can be used for data or code storage and DRAM can be used as a buffer to handle temporary disruptions of the CD data stream.
- ❖ The audio interface units put out video in formats that can be used by A/D converters.
- ❖ General purpose I/O pins can be used to decode buttons, run displays, etc.,
 - ❖ Cirrus provides Reference design for a CD/MP3 player.

Components design and Testing:

- ❖ The audio decompression object can be implemented from existing code or created as new software.
- ❖ In case of an audio system that does not conform to a standard, it may be necessary to create an audio compression program to create test files.

System integration and debugging:

- ❖ System integration and debugging process ensuring that audio plays smoothly and without interruption.
- ❖ Any file access and audio output operations are tested separately using recognizable test signals.

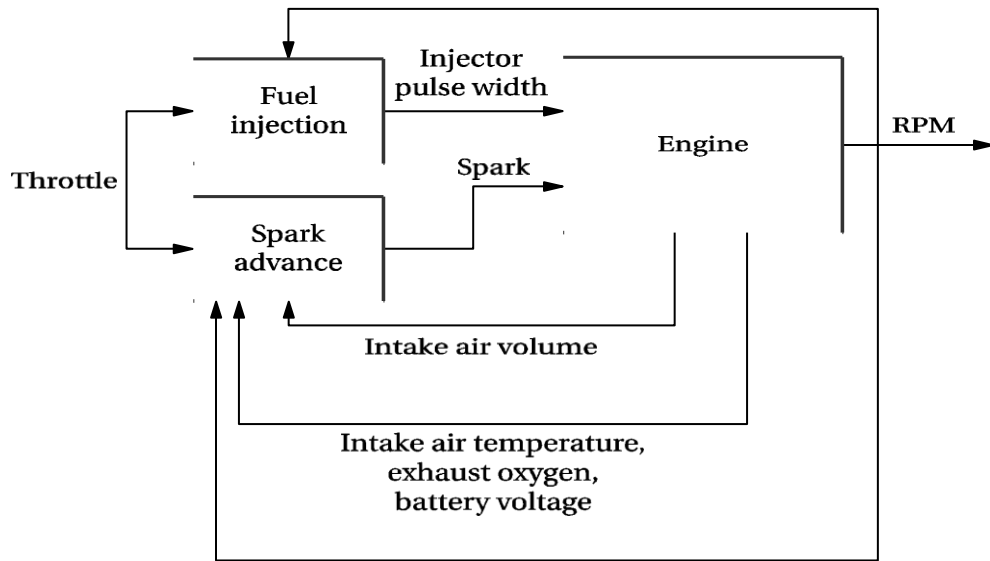
5.9a.4. ENGINE CONTROL UNIT (ECU)

Explain in detail the design of engine control unit (NOV 2017, NOV/DEC 2018, APRIL/MAY 2019)/Multitasking capacity of RTOS helps in engine control automation (DEC 2022/JAN 2023)

- ❖ Engine Control Unit (ECU) is a generic term for any embedded system that controls one or more of the electrical systems or subsystems in a motor vehicle.
- ❖ An Engine Control Unit (ECU) is a type of electric control unit that controls a series of actuators on an internal combustion engine to ensure optimal engine performance.
- ❖ It does this by reading values from a multitude of sensors within the engine body, interpreting the data using multidimensional performance maps called Look-up tables and adjusting the engine actuators accordingly.
- ❖ Engine control unit controls the operation of a fuel-injected engine based on several measurements taken from the running engine.

THEORY OF OPERATION AND REQUIREMENTS

- ❖ Design a basic engine controller for a simple fuel-injected engine. The block diagram of the engine is shown in the figure below.
- ❖ The Throttle is the command input. The engine measures throttle, RPM, intake air volume, and other variables.



The engine controller computes injector pulse width and spark. This doesn't compute all the outputs required by a real engine.

Requirements

Requirements for the engine control unit shown in the below figure.

Name	ECU
Purpose	Engine controller for fuel-injected engine
Inputs	Throttle, RPM, intake air volume, intake manifold pressure
Outputs	Injector pulse width, spark advance angle.
Functions	Compute injector pulse width and spark advance angle as a function of throttle, RPM, intake air volume, intake manifold pressure.
Performance	Injector pulse updated at 2-ms period, spark advance angle updated at 1-ms period.
Manufacturing Cost	Approximately \$50
Power	Power by engine generator
Physical size and weight	Approx. 4 in x 4 in, less than 1 pound

Fig. Requirements for the engine controller

SPECIFICATION

- ❖ The engine controller must deal with processes that happen at different rates. Below figures show the update periods for the different signals.
- ❖ Use N and T to represent the change in RPM and throttle position, respectively. Our controller computes two output signals, injector pulse width PW and spark advance angle S .

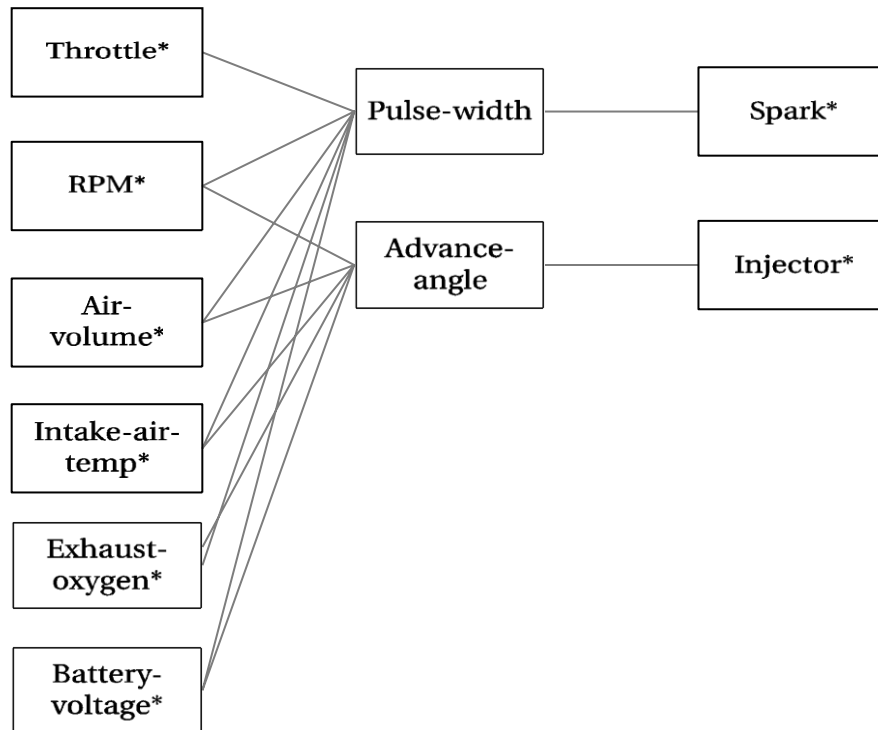
The Controller then applies correction to these initial values:

- As the intake air temperature (THA) increases during the engine warm-up, the controller reduces the injection duration.
- As the throttle opens, the controller temporarily increases the injection frequency.
- The Controller adjusts duration up or down based upon readings from the exhaust oxygen sensor (OX).
- The injection duration is increased as the battery voltage (+B) drops.

Signal	Variable name		
Throttle		input	
		input	
Intake air volume		input	
		output	
		output	
Intake air temperature		input	
Exhaust oxygen		input	
Battery voltage		input	

SYSTEM ARCHITECTURE

Below figure shows the class diagram for the engine controller. The two major process, pulse-width and advance-angle, compute the control parameters for the spark plugs and injectors.

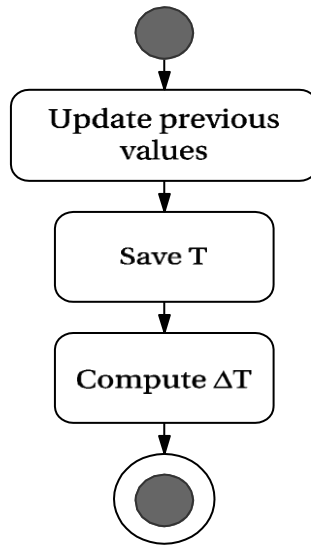


Class diagram for the engine controller

- ❖ The Control Parameters rely on changes in some of the input signals. Use the physical sensor classes to compute these values.
- ❖ Each change must be updated at the variable sampling rate. The update process is simplified by performing it in a task run at the required update rate.

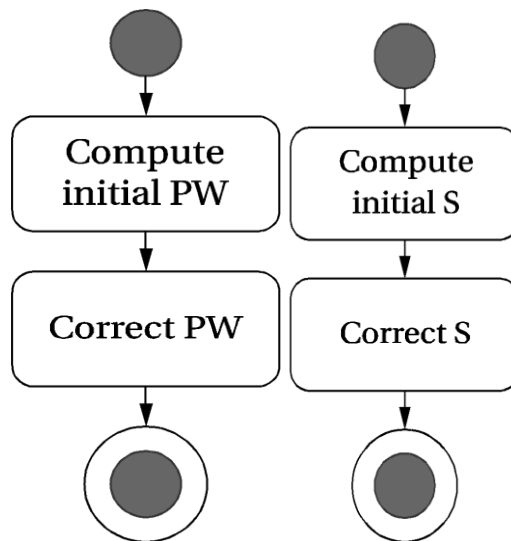
State diagram

- ❖ State diagram for throttle position sensing is shown in below figure. It saves both the current value and change in value of the throttle.
- ❖ Uses similar control flow to compute changes to the other variables.



Throttle position sensing state diagram

Below figure shows the state diagram for injector pulse width and spark advance angle. In each case the value is computed in two stages, first an initial value followed by a correction.



- ❖ The pulse width and advance angle processes do not generate the waveforms to drive the spark and injector waveforms.
- ❖ These waveforms must be carefully timed to the engine's current state.
- ❖ Each spark plug and injector must fire at exactly the right time in the engine cycle, taking into account the engine's current speed as well as the control parameters.

- ❖ Some engine controller platforms provide hardware units that generate high rate, changing waveforms.
- ❖ For example consider MPC5602D. The main processor is a power PC processor. The enhanced modular I/O subsystem provides 28 input and output channels controlled by Timers.
- ❖ Each channel can perform a variety of functions.
- ❖ The output pulse width and frequency modulation buffered mode will automatically generate a waveform whose period and duty cycle can be varied by writing registers in the enhanced modular I/O subsystems.
- ❖ The details of the waveform timing are handled by the output channel hardware.
- ❖ Because these objects must be updated at different rates, their execution will be controlled by an RTOS. Depending on the RTOS Latency, separate the I/O functions into interrupt service handlers and threads.

COMPONENT DESIGN AND TESTING

- ❖ The various tasks must be coded to satisfy the requirements of RTOS processes.
- ❖ Variables that are maintained across task execution must be allocated and saved in appropriate memory locations.
- ❖ The RTOS initialization phase is used to set up the task periods.
- ❖ Because some of the output variables depend on changes in states, these tasks should be tested with multiple input variables sequences to ensure that both the basic and adjustment calculations are performed correctly.

SYSTEM INTEGRATION AND TESTING

Engine generates huge amounts of electrical noise that can cripple digital electronics. They also operate over very wide temperature ranges.

1. Hot during engine operation
2. Very cold before the engine is started.

Any testing performed on an actual engine must be conducted using an engine controller that has been designed to withstand the harsh environment of the engine compartment.

5.9a.5.VIDEOACCELERATOR

Discuss in detail about embedded concepts in the design of video accelerator

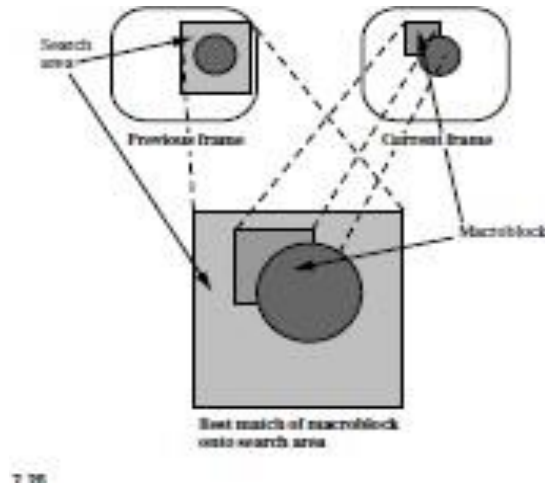
(8 Marks) Nov/Dec 2016 or Illustrate the working of video player (NOV/DEC 2018) or Illustrate video accelerator using UML methodology (NOV/DEC 2018) (April/May 2019) (Dec 2022/Jan 2023) (April/May 2023)

A video accelerator is a hardware circuit on a display adapter that speed up full motion video, which also frees the CPU to take care of other tasks. Motion estimation engines are used in real-time search engines; we may want to have one attached to our personal computer to experiment with video processing techniques.

Algorithm and Requirements

- Block motion estimation is used in digital video compression algorithms so that one frame in the video can be described in terms of the differences between it and another frame.
- Because objects in the frame often move relatively little, describing one frame in terms of another greatly reduces the number of bits.
- The goal is to perform a two-dimensional correlation to find the best match between regions in the two frames.
- We divide the current frame into *macroblocks*.
- We want to find the region in the previous frame that most closely matches the macroblock.
- Searching over the entire previous frame would be too expensive, so we usually limit the search to a given area, centered around the macroblock and larger than the macroblock.
- Intensity is measured as an 8-bit luminance that represents a monochrome pixel—color information is not used in motion estimation.
- We choose the macroblock position relative to the search area that gives us the smallest value for the metric.

- The offset at this chosen position describes a vector from the search area center to the macroblock's center that is called the *motion vector*.
- For simplicity, we will build an engine for a full search, which compares the macroblock and search area at every possible point.



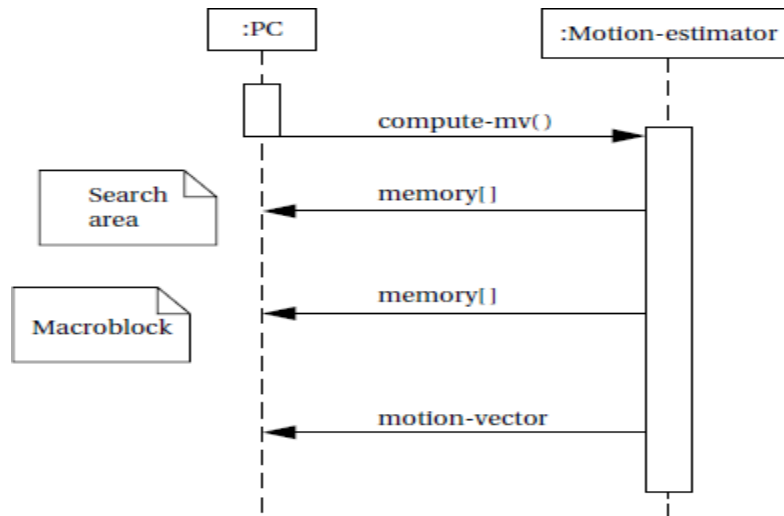
Requirements:

Name:	Block motion estimator
Purpose:	Perform block motion estimation within a PC system
Inputs:	Macroblocks and search areas
Outputs:	Motion vectors
Functions:	Compute motion vectors using full search algorithms
Performance:	As fast as we can get
Manufacturing cost:	Hundreds of dollars
Power:	Powered by PC power supply
Physical size:	Packaged as a PC card for PC

Specification

- The specification for the system is relatively straightforward because the algorithm is simple.
- Because the behavior is simple, we need to define only two classes to describe it: the accelerator itself and the PC.

- The PC makes its memory accessible to the accelerator.
- The accelerator provides a behavior `compute-mv()` that performs the block motion estimation algorithm.
- After initiating the behavior, the accelerator reads the search area and macroblock from the PC; after computing the motion vector, it returns it to the PC.



Sequencediagramofaccelerator

Architecture

- The accelerator will be implemented in an FPGA on a card connected to a PC's PCI slot.
- Such accelerators can be purchased or they can be designed from scratch.
- If you design such a card from scratch, you have to decide early on whether the card will be used only for this video accelerator or if it should be made general enough to support other applications as well.

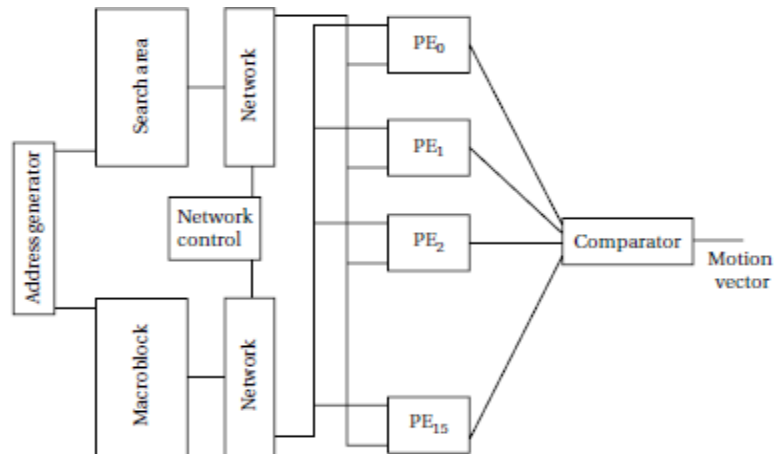


FIGURE 7.31
An architecture for the motion estimation accelerator [Dut96].

Component Design

- If we want to use a standard FPGA accelerator board to implement the accelerator, we must first make sure that it provides the proper memory required for M and S .
- Once we have verified that the accelerator board has the required structure, we can concentrate on designing the FPGA logic.
- Designing an FPGA is, for the most part, a straightforward exercise in logic design. Because the logic for the accelerator is very regular, we can improve the FPGA's clock rate by properly placing the logic in the FPGA to reduce wire lengths.
- If we are designing our own accelerator board, we have to design both the video accelerator design proper and the interface to the PCI bus.
- We can create and exercise the video accelerator architecture in a hardware description language like VHDL or Verilog and simulate its operation.
- Designing the PCI interface requires somewhat different techniques since we may not have a simulation model for a PCI

System Testing

- Testing video algorithms requires a large amount of data. Luckily, the data represents images and video, which are plentiful.

- Because we are designing only a motion estimation accelerator and not a complete video compressor, it is probably easiest to use images, not video, for test data.
- You can use standard video tools to extract a few frames from a digitized video and store them in JPEG format.
- Open [source for JPEG encoders and decoders](#) is available. These programs can be modified to read JPEG images and put out pixels in the format required by your accelerator.
- With a little more cleverness, the resulting motion vector can be written back onto the image for a visual confirmation of the result. If you want to be adventurous and try motion estimation on video, open source MPEG encoders and decoders are also available

