Introduction to Real Time Systems

What is a Real Time System ?

- There is no single, all embracing definition of what constitutes a real-time system.
- Those definitions that do exist should be viewed more as a guide to the reader in determining if they are dealing with a real-time system or not.
- This is not such a contradiction as it might sound, since the term ‘real-time’ is more a concept that means many different things to different people than a set of hard definitions.
- In fact it is equally difficult to define what constitutes a non real-time system
- As such you may find that some aspects of the more popular definitions may well apply to situations that would be classed as non real-time.
- Furthermore a real-time system does not have to exhibit all of these definitions to be classified as real time in fact some real-time systems may exhibit behaviour that acts **contrary** to one or more of these definitions, but **agree** with others.
- Taking all this into account, the most popular Real-time definitions in increasing order of priority are listed below.

Introduction to Real-Time Systems

Popular Real-Time System Definitions

. . . *"A real time system is a controlling system taking in information from its environment, processing it and generating a response to it."*

. . . *" A real time system reacts, responds and alters its actions so as to affect the environment in which it is placed."*

. . . *"A real time system implies some air of criticality implied by its response time.*

. . . *"A real-time systems response to events do not have to mean fast, they just have to be 'timely', the definition of which will vary from one system to another. It could vary from uSec to minutes".*

. . . *"A real time system is one where the correct answer at the wrong time is the wrong answer"*

Calculable

. . . *"A real time system has a guaranteed, calculatable (we use the word deterministic) , worst case response time to an event under its control.*

Classification of Real-time systems - Hard vs. Soft

Hard Real Time Systems.

- A hard real-time system is one in which a failure to meet a specified response time results in overall system failure.
- A hard real-time system will have a specified maximum delay to a response, which can then be used to judge failure.

Example Hard Real Time Systems

  - Fuel injection management system in a car.
  - Nuclear Power Station Control Rod Activation.
  - A Railway level crossing system failing to detect a train approaching in time.
  - Manufacturing Robot, assembling components.

- In other words, failure of a hard real-time system usually results in some catastrophic failure of the system perhaps resulting in loss of life, serious injury or damage.
- In other words hard real time systems are ‘time critical’.

Introduction to Real-Time Systems

Soft Real Time Systems

- In contrast to a hard real-time system, a soft real-time system implies that failure to meet a specified response time merely results in system degradation not necessarily outright failure.
  - *"...A Soft Real time system thus quotes a typical, suggested or average response time against which degradation can be judged.*
  - *"…The response time of a Soft Real time system is thus not fixed and may improve or degrade depending upon system loading*"
- Of course system degradation can ultimately becomes system failure if the response time becomes intolerable.

Example Soft Real Time Systems

- An Elevator control system. This could be said to have failed if the requester decides to walk instead.
- Cash dispenser or ATM for a bank. This could be said to have failed if the transaction cannot be completed without annoying the customer.
- Video Arcade Game. This could be said to have failed when the game becomes so unresponsive that the player looses the game.

Further Classification of Real-Time Systems

- Real time systems can also be classified on the basis of the way in which they generate a response to an event. There are two classifications: -
  - Event driven systems
  - Time driven systems

Event Driven Real-Time systems.

- An event driven system, is one in which a sensor is responsible for detecting that some event or parameter critical to the systems operation has taken place or changed. This information is relayed to the system either an 'interrupt' or it can be detected by polling in software.
- The occurrence of the event thus causes the system to temporarily suspend its current activity, (unless that activity has a higher priority than generating the response to the sensor) while it generates a response to it.

Advantages of Interrupt Based Event Driven Systems:-

- Events should never be missed and can even be buffered up to be dealt with later if the system is busy. For example typing ahead on a keyboard. This is in contrast to a polled approach whereby if the system is too slow in executing its polling loop it may miss an event altogether.
- The system is able to carry out more mundane processing at a lower priority level, or even sit idle, while waiting for the event to inform it of the need to respond.
- Events can be prioritised in hardware based on their importance to the integrity of the system, thus a high priority event can override a lower priority one.
- Adding more event sensors to the system should not affect the response time to each one of them individually, provided the system is not overloaded with events or work.

Disadvantages to Interrupt Based, Event Driven Systems

- Usually require more complex/expensive hardware and software architectures. For example the system may require
    - Additional hardware to prioritise Interrupt requests from sensors
    - Sensors capable of generating an interrupt
    - Operating system device driver to deal with the interrupt and route the event to a users program
- More difficult to debug.
    - Interrupts are asynchronous to the operation of the rest of system and can occur with unpredictable frequency and timing relative to the execution of the program, making it very difficult to single step or breakpoint such a system in order to trace a bug dependent upon that set of events.

Advantages and Disadvantages to Polled, Event Driven Systems

- Advantage: Software is easier to write and test since there are no asynchronous interrupts that can be generated in the system and thus there is no complex interrupt service routine (ISR) or device driver with which the system has to integrate with.

- Advantage: Systems can more easily be single stepped and break-pointed to aid the debugging process and are easier to exhaustively test since events can only occur at recognised points in the program execution (i.e. synchronous to the program execution)

- Disadvantage: There is a possibility that a sensor event could be missed, if the stimulus is too brief to be recognised within the polling period.

  N.B. You could probably arrange for additional hardware to latch the event ensuring that it will ultimately be detected, but there is no guarantee that the system would detect the stimulus quickly enough when polling if it is distracted or busy elsewhere.

- Disadvantage: Adding more sensors to the system will, on average, degrade the response time since it takes longer to poll all the devices.

- Disadvantage: It is easy for the polling process to take longer than is required to generate the response

- Disadvantage: Polling is very time consuming, wasting CPU resources that could be better employed elsewhere and by necessity ceases to work if the CPU is occupied by other things

Time Driven Real-Time Systems

- In time driven systems, the system is responsible for ensuring that certain responses or activities occur at specified times or at regular intervals.
- Usually, a hardware timer within the computer can be used to generate the timing information for such activities and can be tuned for the resolution required.
- For example, in a process control situation, there may be several sensors and several activities, e.g. A, B and C. The system may be required to perform activity
  - A every 50mS,
  - B every 30mS
  - C every 100mS.
- In such a system, the designer would generally decompose the system activities into a number of smaller threads or processes (see next section) and ensure that the operating system scheduler ran the tasks at the prescribed times (assuming that this is possible, we will come back to this when we discuss Scheduling strategies later in the course)