Distributed & Embedded Real-Time Systems (0640751)

Lecture (7)

DERTS Design Requirements (3): System Design Approach

Prof. Kasim M. Al-Aubidy
Philadelphia University

Lecture Outline:

- ERTS design: SoC and SoB.
- RT Embedded Technologies.
- System Features with RTOS.
- Why we use a RTOS?
- DERTS Programming Languages.
- Software Design.
- Describe and explain by examples the basic task synchronization mechanisms.

Embedded System Design:

ERT system is dealing with solving – real time constrains:

- **Real time Response**
  - The system needs an immediate response – can even be in magnitude order nano-seconds
  - Asynchronous events can occur at any time.

- **Race Conditions and Timing**
  - Buffers zone – limited size needs timed treatment.
  - Mutual demand for resources.

Embedded System Design:

- **Size limitation:**

– System on Chip (SoC) or System on Board (SoB) tends to be very limited in space.

➢This forces many limitation on the processor in terms of address lines number etc …

- **Power Consumption:**

– The power budget is very limited and extremely important especially in mobile applications

Embedded System Design:

- **Performance:**

– The system performance is a key issue and must be kept in all Circumstance.

– Performance is a major factor in choosing the processor, the clock frequency , ram size , code size etc..

- **Re-use:**

– In many projects the system relies on already developed components, which the designer must reuse. These components already have embedded constrains in addition to the new system constraints.

- **Recovering from Failures:**

– Working in a distributed environment – connection failure.

Real Time Design:

DERT system design concerns the following:

– Is the **architecture** suitable?

– Are the **link speeds** adequate?

– Are the **processing components** powerful enough?

– Is the **Operating System** suitable?

– What is the **footprint - size** of the code ?

– What is the required **size** of the **RAM** ?

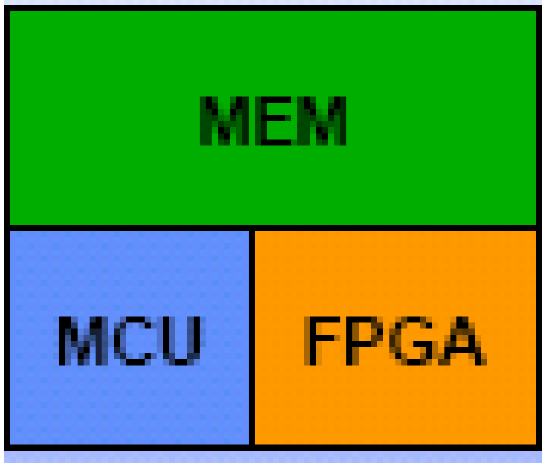
– Could we keep the **power budget** ?

– What are the **tools this processor** has - the IDE ?

System On Chip (SoC):

The System On Chip consists of a few different block types inside the same chip like:

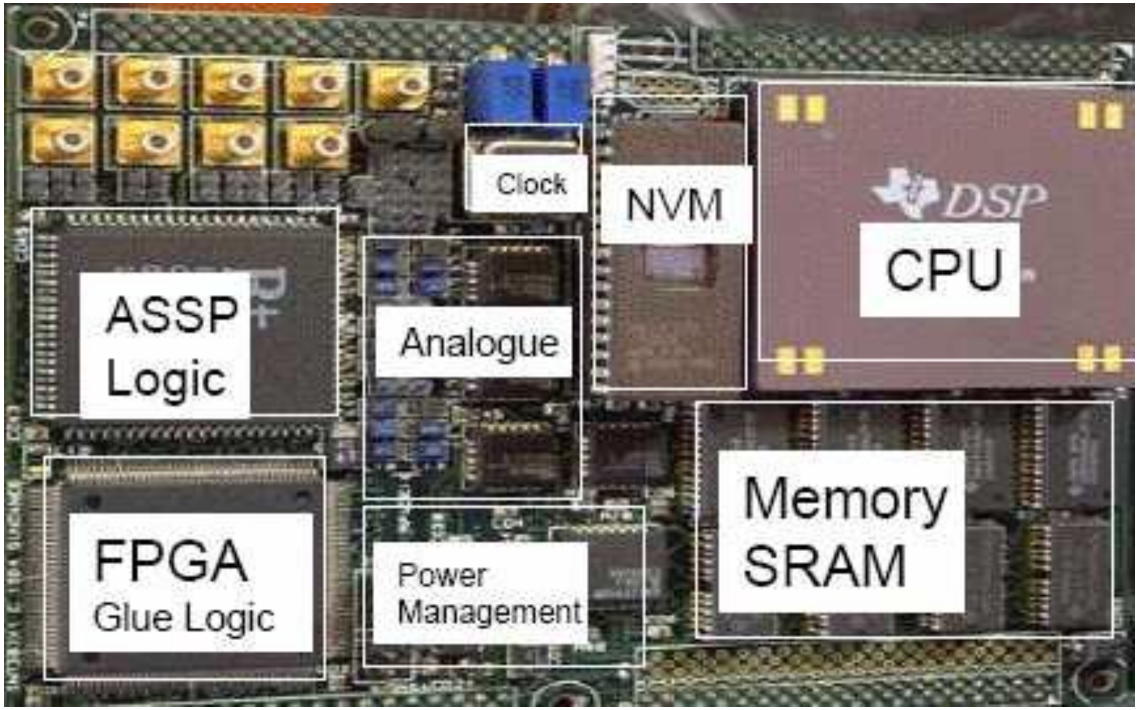
- CPU: The main processor unit
- Memory devices: Volatile and non-volatile
- Internal bus interfaces and decoders
- Peripheral devices
- Digital/Analog functions



System On Board (SoB):

Full set of the necessary devices as standalone on the same board:

- CPU: The main processor unit
- Memory devices: Volatile and non-volatile
- Internal bus interfaces and decoders
- Peripheral devices



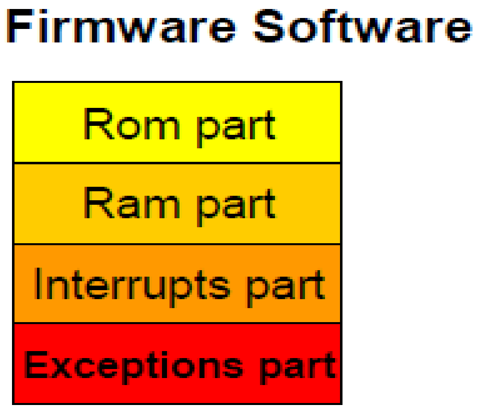
RT Embedded Technologies:

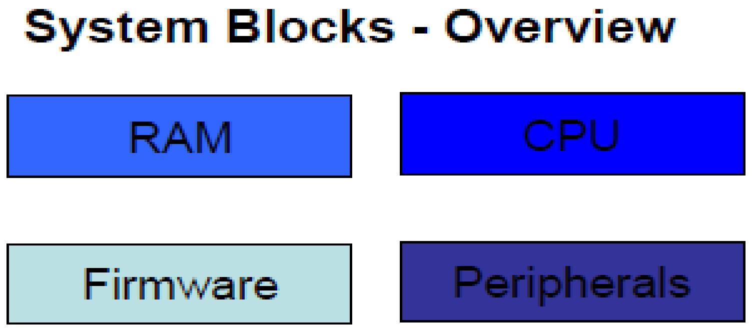
1. **Platforms types:**
    - Firmware (SoC/SoB)
    - Real-Time Operating System (RTOS)
2. **Drivers technology.**

The firmware systems

➢ About 25% of the system are without OS.

➢ The OS mechanism is not a must for this kind of usually simple applications.





Firmware Systems:

The main features of Firmware systems are;

- Small code size: (5-100k) per processor.
- Using only procedure level programming.
- Main Loop system.
- Drivers for communication and/or peripheral devices can hold a major part of the system.
- The firmware can manage and control a single SOC, or can control also devices on a full Board level.

System with RTOS:

The main features of systems with RTOS are;

- *Medium & Large code size (100k - 4m) per processor.*
- *A system can hold several processors.*
- *Using all programming levels.*
- *RTOS includes :*
  - time base task scheduling
  - mutual exclusion treatment
  - task priority
  - etc…

What is an Operating System:

An Operating System (OS) is a computer program that manages the hardware and software resources of a computer

Operating system performs basic tasks such as:

- – Controlling and allocating memory
- – Prioritizing system requests
- – Controlling input and output devices
- – Facilitating networking
- – Managing files

➢ In other words, it forms a platform for other software.

DERT System with RTOS:

➢ Drivers for communication and/or peripheral devices usually hold a minor part of the system.

➢ The system with RTOS can reside on a system on chip (SOC) or on a System On Board (SOB) at a full Board level

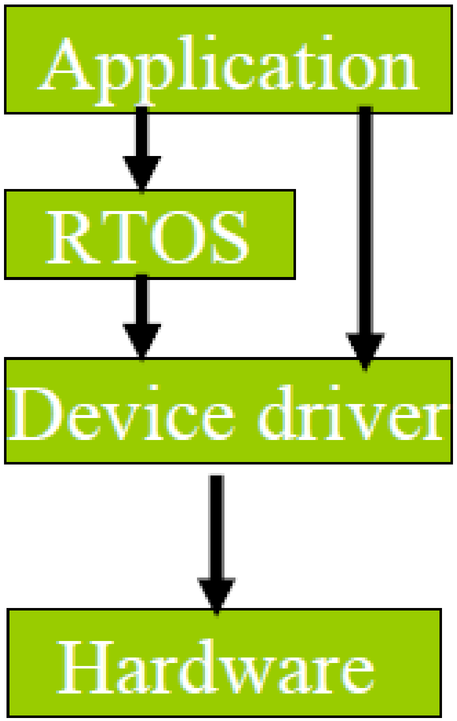
RT Embedded Technologies

Device Drivers:

• What are device drivers?

– Make the attached device work.

– Insulate the complexities involved in I/O handling.



Device Drivers' Functionalities:

– Initialization

– Data access

– Data assignment

– Interrupt handling

DERT System Programming Languages:

Old Systems: include many parts in assembly language; processor dependent language.

Current Systems: most real time systems are based on RTOS now, written in the C language. Still, for crucial time tasks, Assembly code may be required (e.g when cycle count is important).

Notes:

• **Some Applications** which have a lot of software, less real-time oriented , but application business dependent , or which inherited software base - **are written using C++.**

• Many applications have several programming languages mixed due to real-time constrains or inherited code.

Languages for DERT Systems:

1. Assembly Language:

- The assembly language is processor dependent.
- No reuse to other processors.
- Mastered by a limited number of programmers.
- Has all the known limitation of assembler:
  - One C line – on average 5 assembly commands.
  - Very hard to check and debug.
- Very efficient in terms of execution time and code size.

Languages for DERT Systems:

2. The C Language:

– Is NOT processor dependent.
– Reuse to other processor is very common.
– Known to a very large number of programmers.
– Relatively easy to check and debug.
– Quite efficient in terms of execution time and code size.
– The compiler can include optimizers for using the processor pipeline structure efficiently.

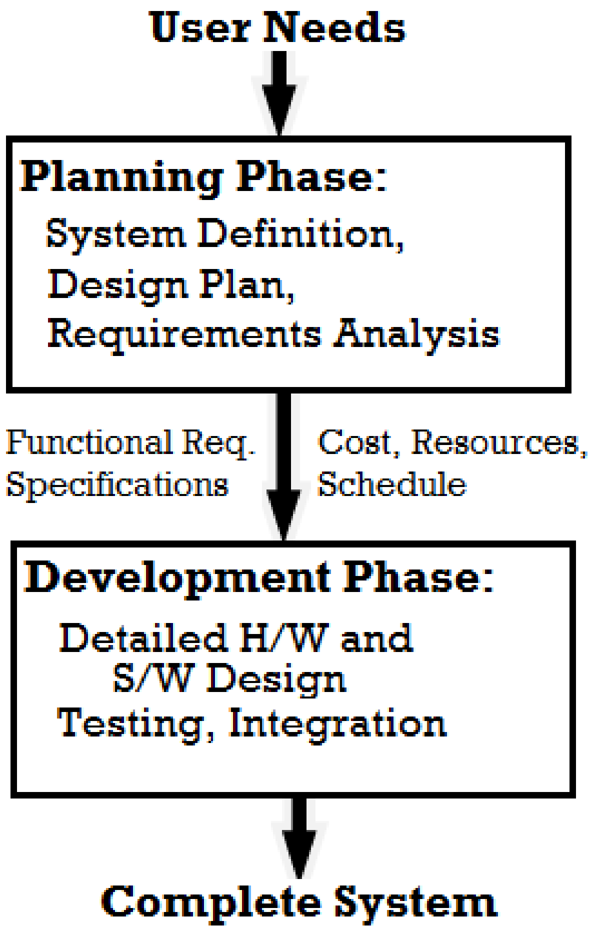
3. The C++ Language:

– C++ language is NOT processor dependent.
– Reuse to other processor is extremely common.
– Known to a very large number of programmers.
– Medium effort require for testing and debug.
– NOT efficient in terms of execution time and code size.

Real-Time Systems Design:

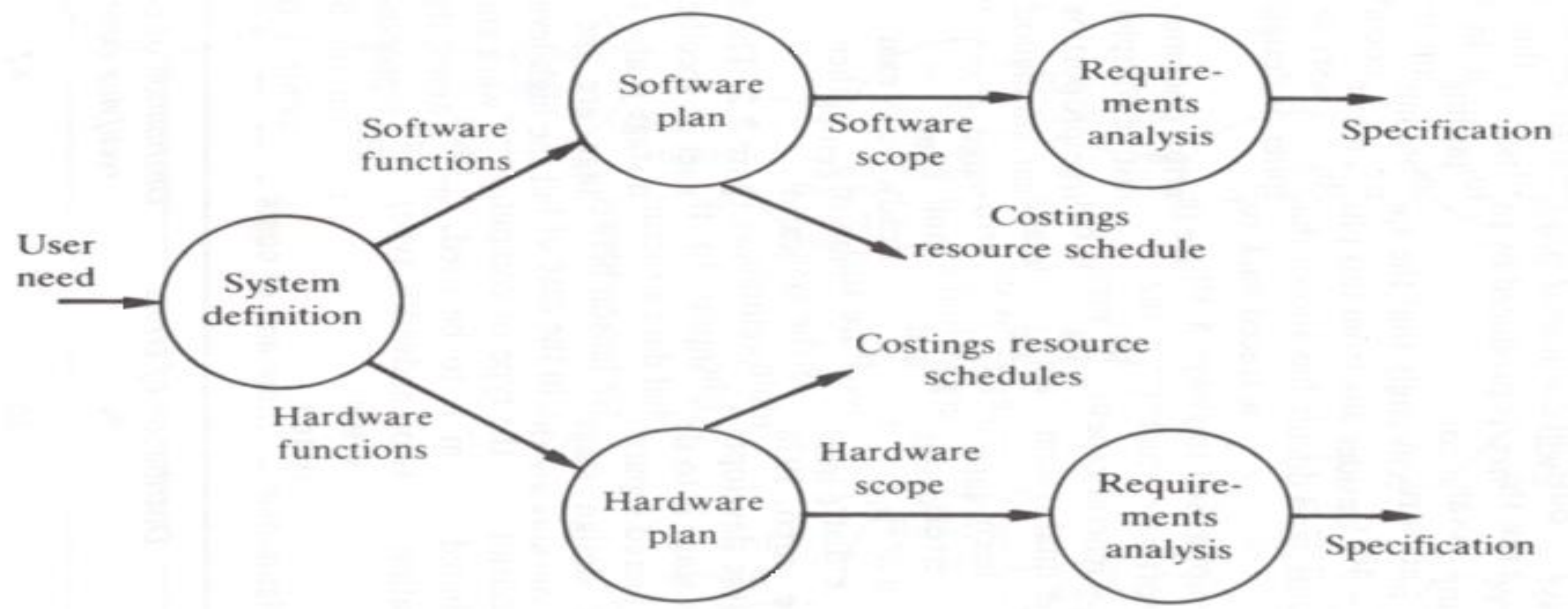
The approach to the design of RTSs is no different in outline from that required for any DERT system. The work can be divided into two main sections:

1. The planning phase, and
2. The development phase.



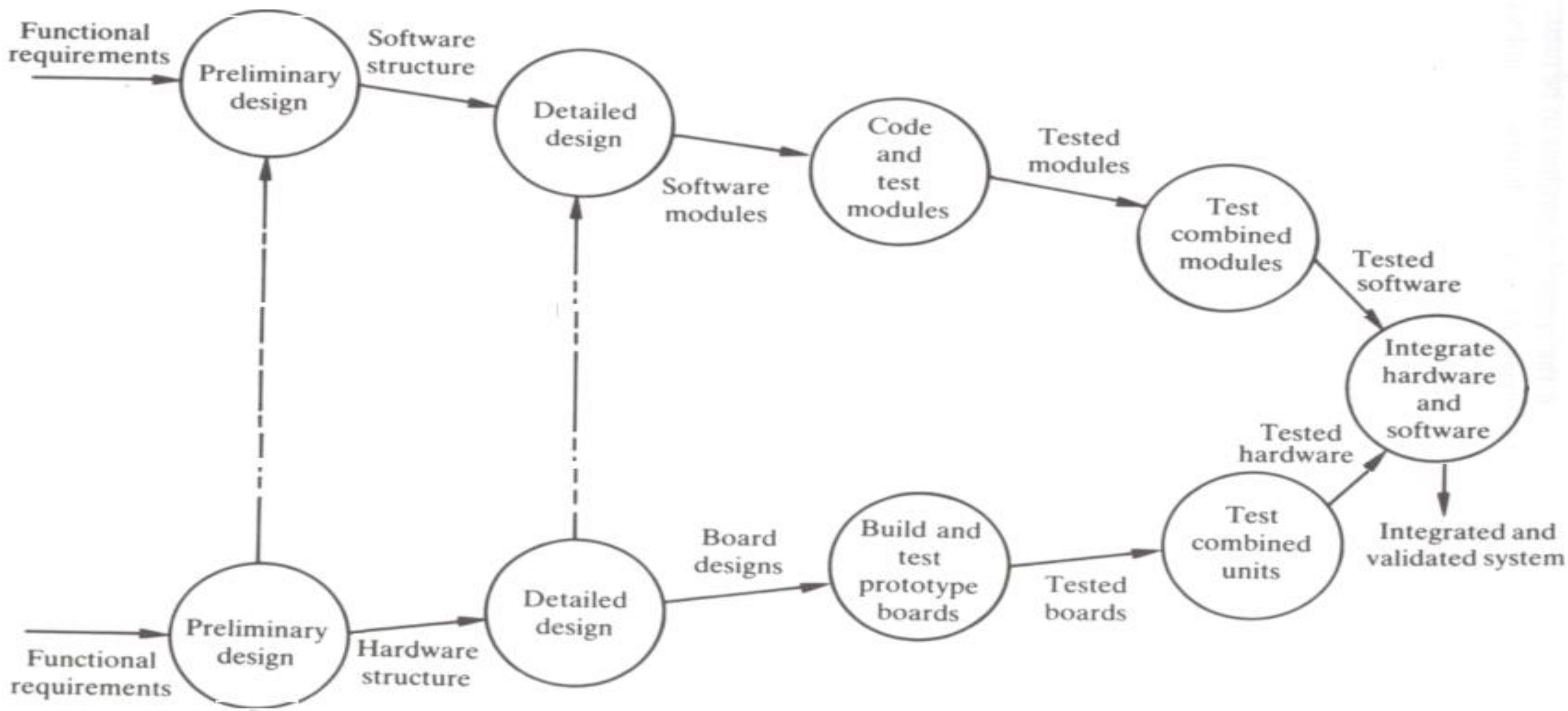
1. Planning Phase:

- It is concerned with interpreting user requirements to produce a detailed specification of the system to be developed and outline plan of the resources, people, time, equipment and costs.
- At this stage preliminary decisions regarding the division of functions between hardware and software will be made.



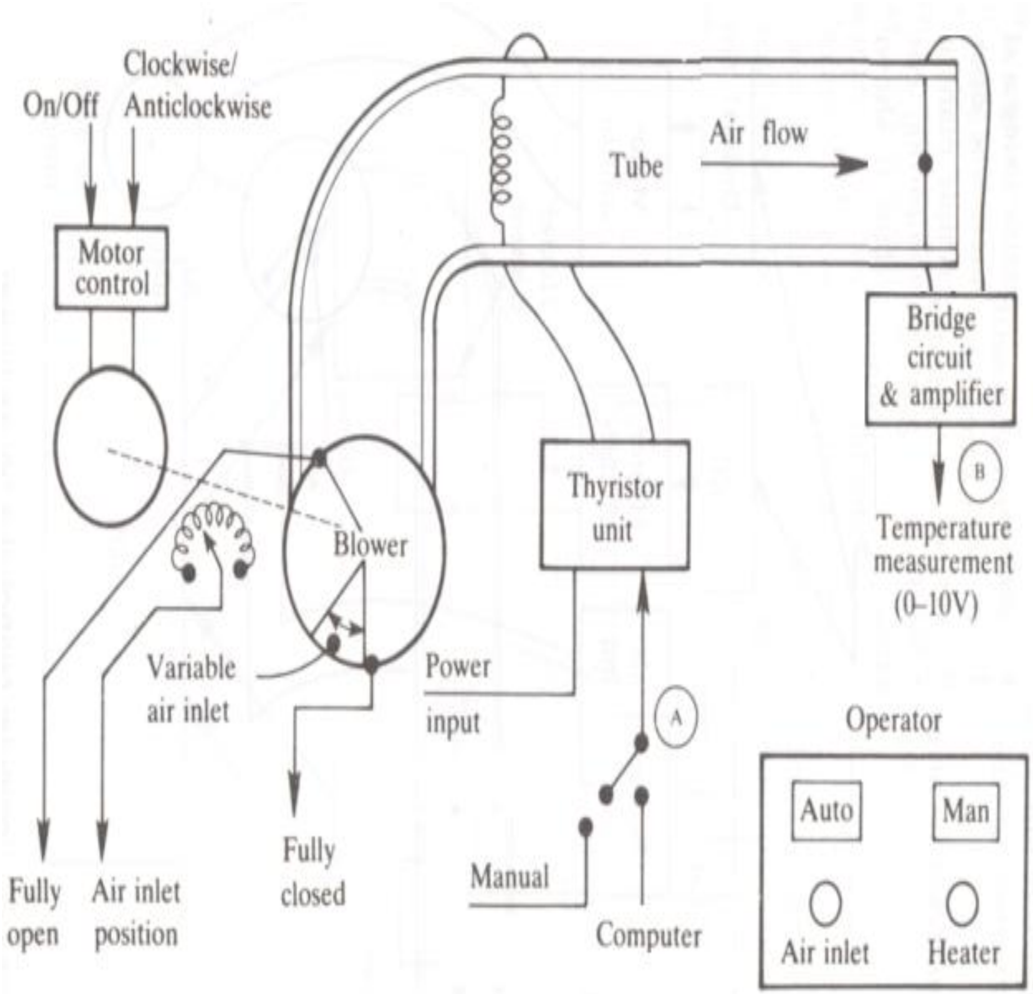
2. Development Phase:

- The inputs to this stage are the HL specifications. During this stage extensive liaison between hardware and software designers is needed.
- The detailed design is usually broken down into two stages;
  - Decomposition into modules, and
  - Module internal design.



Specification Document:

Example: Hot-air blowers.



Display

The operator display is as shown below:

Set temperature	:nn.n °C	Date	:dd/mm/yyyy
Actual temperature	:nn.n °C	Time	:hh.mm
Error	:nn.n °C		
Heater output	:nn% FS	Sampling Interval	:nn ms
Controller settings			
Proportional gain	:nn.n		
Integral action	:nn.nn s		
Derivative action	:nn.nn s		

The values on the display will be updated every 5 seconds.

Operator input

1. Set temperature = nn.n
2. Proportional gain = nn.n
3. Integral action = nn.nn
4. Derivative action = nn.nn
5. Sampling interval = nn
6. Management information
7. Accept entries

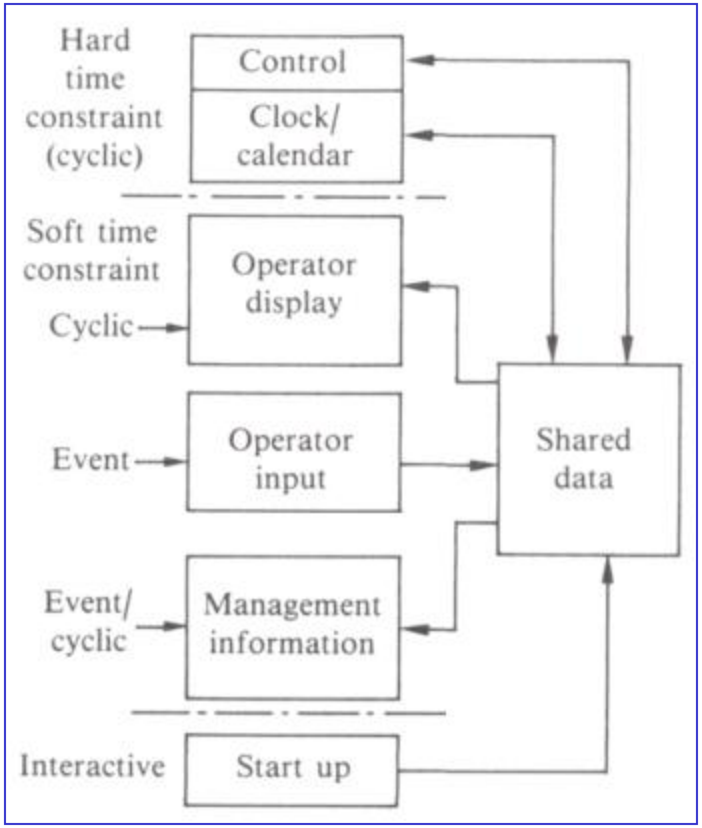
Select menu number to change

Preliminary Design:

Hardware Design: to be discussed in lecture.

Software Design:

- The required software must perform several functions:
  - DDC for temperature control.
  - Operator display.
  - Operator input.
  - Management information.
  - System start-up and shut-down.
  - Clock/calendar function.
- The control module has a hard constraint, it must run every 40 msec.
- The clock/calendar module must run every 20 msec.
- The operator display has a hard constraint in that an update interval of 5 sec is given.
- Soft constraints are adequate for operator i/p and for the management information.



Software Design:

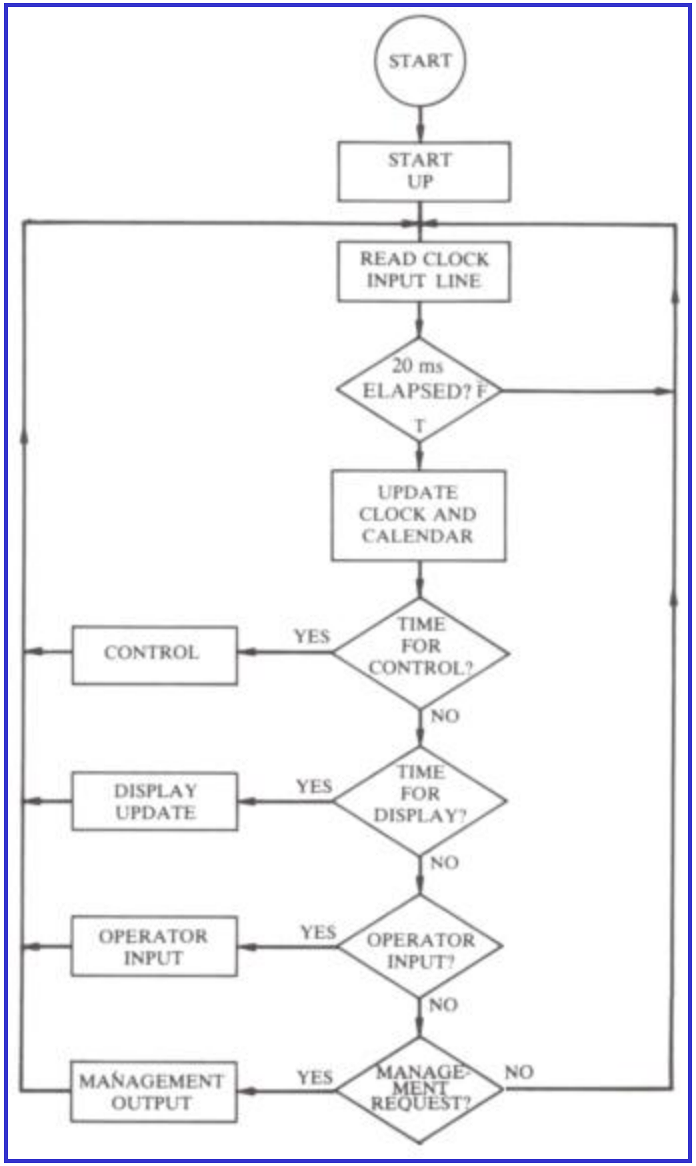
- There are several different activities which can be divided into sub-problems. The sub-problems will have to share information. To achieve this, there are three approaches:

1. Single-Program Approach:

- The modules are treated as procedures or subroutines of a single program.
- This structure is easy to program, however, it imposes the most severe of the time constraints.

Example: for the system to work the clock/calendar module and any one of the other modules must complete their operations within T. t_1, t_2, t_3, t_4 and t_5 are the maximum execution times for the given modules, then a requirement for the system to work is;

$$t_1 + \max(t_2, t_3, t_4, t_5) \angle T$$

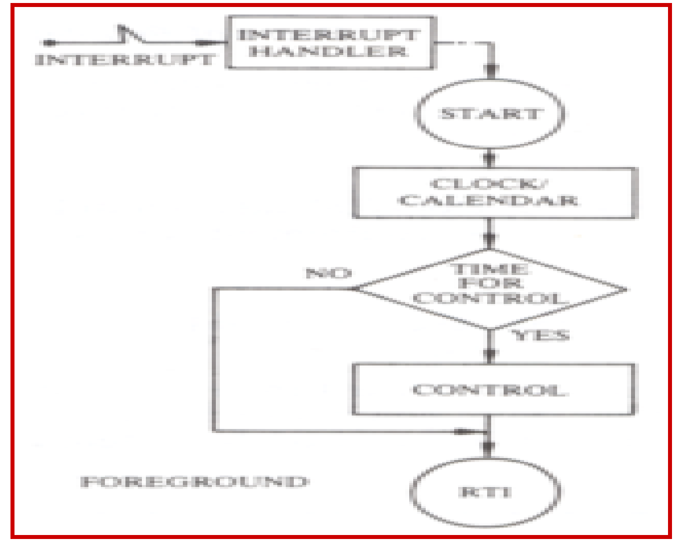


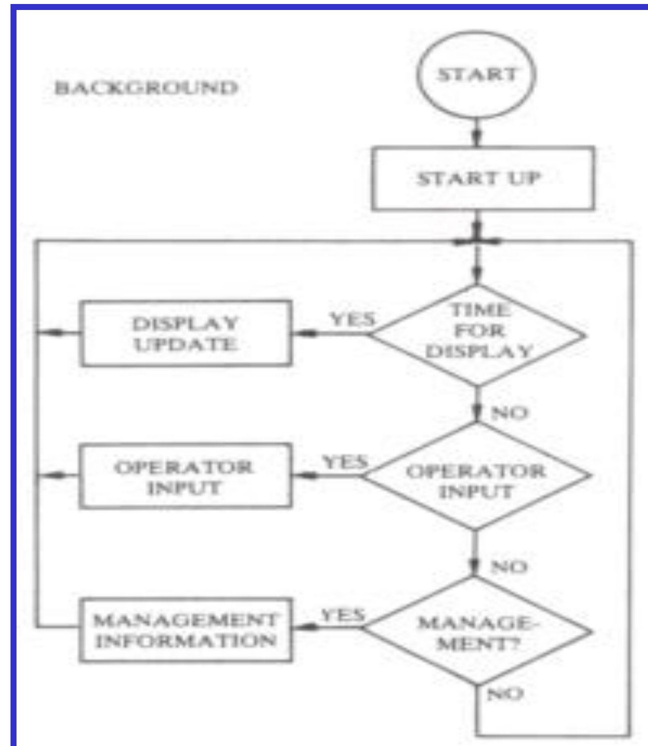
2. Foreground/Background System:

- There are advantages (less time constraints) if the modules with hard time can be separated from, and handled independently of, the modules with soft time constraints or on constraints.
- The modules with hard time constraints are run in “foreground” and the modules with soft constraints (or no constraints) are run in the “background”.
- The partitioning into foreground and background usually requires the support of a real-time OS.
- A requirement for the foreground part to work is that:

$$t_1 + t_2 \angle T$$

- A requirement for the background part to work is that;
- max(t_3, t_4, t_5) is less than 10 sec.
- Display module runs on average every 5 sec, and
- Operator input responds in less than 10 sec.



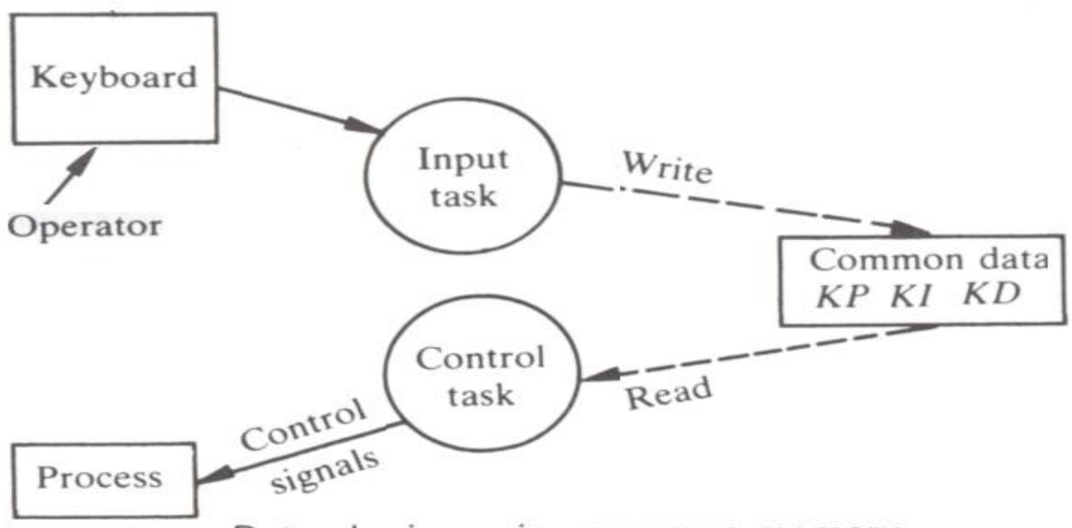


3. Multi-Tasking Approach:

- The design and programming of large RT systems is eased if the foreground/background partitioning can be extended into multiple partitions to allow the concept of many active tasks each can be carried out in parallel.
- The implementation of a multi-tasking system requires the ability to;
  - Create separate tasks.
  - Schedule running of the tasks on a priority basis.
  - Share data between tasks.
  - Synchronize tasks with each other and with external events.
  - Prevent tasks corrupting each other.
  - Control the starting and stopping of tasks.
- The facilities to perform the above actions are typically provided by a RTOS or a combination of RTOS and a real-time programming language.

Mutual Exclusion:

- Consider the transfer of information from i/p task to a control task. The i/p task gets the values for the controller i/p parameters (gain, Ti and Td). From these it computes the controller parameters (KP, KI, and KD) and these are transferred to the CONTROL task.
- A simple method is to hold the parameters values in an area of memory (common data area) and hence is accessible to both tasks.



Data sharing using common memory

For more information:

1. http://www.cs.ou.edu/~fagg/umass/classes/377f02/lectures.html
2. http://www.cs.umbc.edu/~younis/Real-Time/CMSC691S.htm#D