Distributed & Embedded Real-Time Systems (0640751)

Lecture (8)

DERTS Design Requirements (4): Software Requirements

Prof. Kasim M. Al-Aubidy
Philadelphia University

Lecture Outline:

- Software Design.
- Program Development Tools.
- Pseudocode programming.
- Synchronization Methods
- Describe and explain by examples the basic task synchronization mechanisms.

Microcontroller Programming:

- MCs have traditionally been programmed using the assembly language of the target device. As a result, the assembly languages of the microcontrollers manufactured by different firms are totally different and the user has to learn a new language before being able program a new type of device.
- MCs can be programmed using high level languages such as BASIC, and C.

HLL Advantages:

High-level languages offer several advantages compared to the assembly language:

It is easier to develop programs using a high-level language.

- Program maintenance is much easier if the program is developed using a high-level language.
- Testing a program developed in a high-level language is much easier.
- High-level languages are more user-friendly and less prone to making errors.
- It is easier to document a program developed using a high-level language.

HLL Disadvantages:

- The length of the code in memory is usually larger when a high-level language is used.
- The programs developed using the assembly language usually run faster than those developed
- using a high-level language.

Software Requirements for DERT Systems:

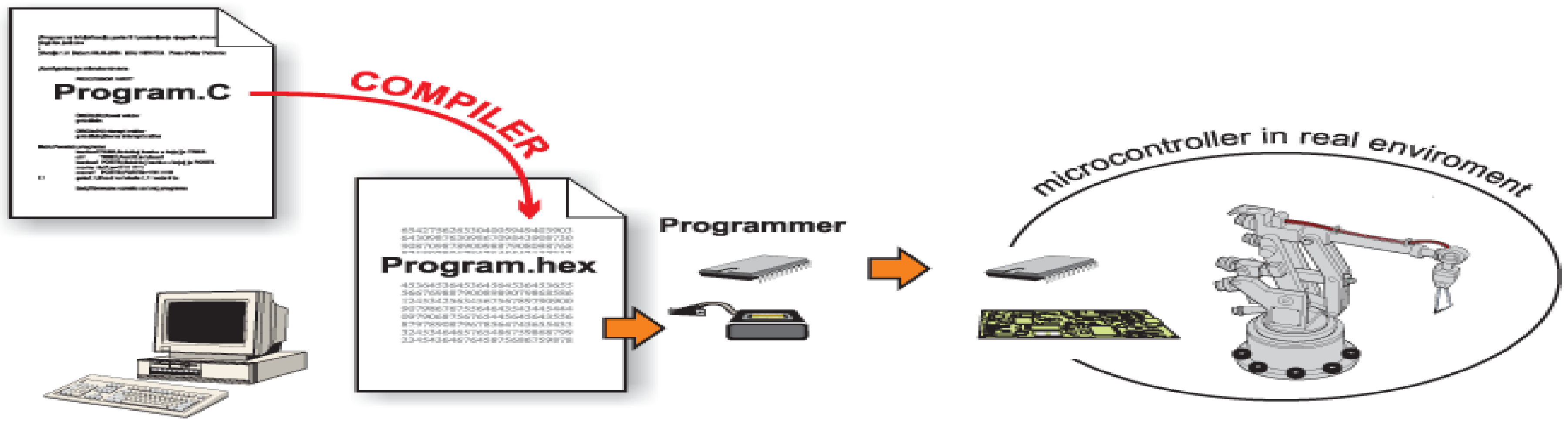
- Computer hardware is nowadays very fast, and control computers are generally programmed using a high-level language. The use of the assembly language is reserved for very special and time-critical applications, such as fast, real-time device drivers.
- C is a popular language used in most computer control applications. It is a powerful language that enables the programmer to perform low-level operations, without the need to use the assembly language.
- The software requirements in DERT systems are as follows:
  - the ability to read data from input ports;
  - the ability to send data to output ports;
  - internal data transfer and mathematical operations;
  - timer interrupt facilities for timing the controller algorithm.

Software Requirements:

• The following software products will be required during system development;

- Program development software (editor): to write the program code.
- MC assembler or compiler: to produce object code.
- MC device programmer software: to transfer the object code to the program memory of the MC.

• Depending on the complexity of the project, additional software products, such as simulators, debuggers or in-circuit emulators, can be used to test and verify the operation of a program.



Program Development Tools:

1. **Flowcharts:** are only useful for small applications.

Disadvantages of flow charts:

- The drawing and modifying the diagrams can be very time-consuming.
- They tend to produce unstructured code which is very difficult to maintain.

2. Structure charts: are similar to flow charts but are easier to draw and modify. Structure charts also tend to produce well-structured code which is easy to understand and maintain.

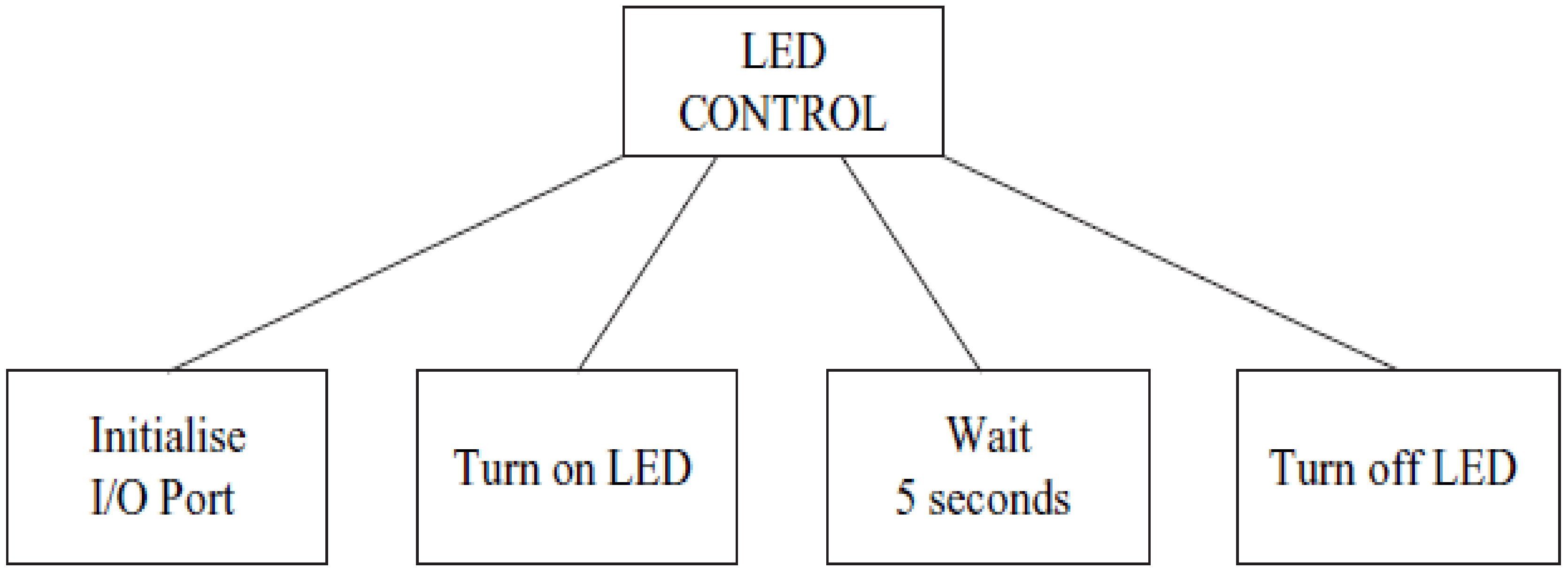
The three basic operations of **sequence**, **selection** and **iteration** are shown differently using structure.

Sequencing using structure charts:

- Sequence is shown with rectangles drawn next to each other.
- The sequence of operations is from left to right.

Example:

First the I/O port is initialized, then the LED is turned on, and finally the LED is turned off after a 5 s delay.

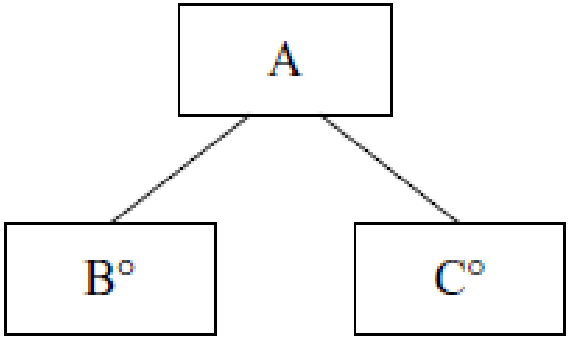


Selection using structure charts:

Selection is shown by placing a small circle at the top right-hand side of a rectangle.

Example:

if *condition1 is true then process B is performed, and if condition2 is true process C is performed.*

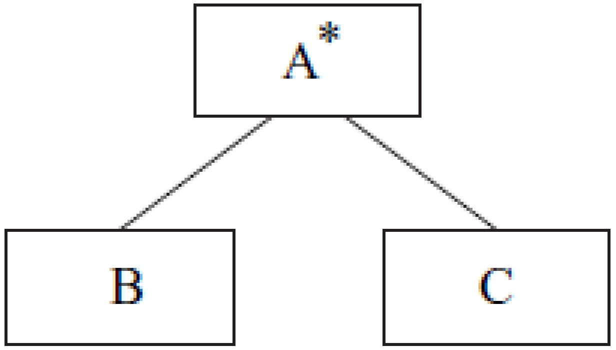


Iteration using structure charts:

Iteration is shown by placing an asterisk sign at the top right-hand side of a rectangle.

Example:

Processes B and C are repeated.



3. Pseudocode Method:

- Pseudocode is a kind of structured English for describing the operation of algorithms.
- It allows the programmer to concentrate on the development of the algorithm independent of the details of the target language.
- It is based on the concept that any program consists of three major items: sequencing, selection, and iteration.
- It is then developed using English sentences to describe algorithms, and this code cannot be compiled.

BEGIN–END

This construct is used to declare the beginning and end of a program or module. Keywords such as ':MAIN' can be used before BEGIN to declare the beginning of the main program:

```
:MAIN
BEGIN
  . . .
  . . .
END
```

```
:ADD
BEGIN
  . . .
  . . .
END
```

Sequencing:

- A sequence is a linear progression where the tasks are performed sequentially one after the other.
- Each action should be written on a new line and all the actions should be aligned with the same indent.
- These keywords can be used for the description of the algorithm:

  Input: READ, GET, OBTAIN

  Output: SEND, PRINT, DISPLAY, SHOW

  Initialize: SET, CLEAR, INITIALIZE

  Compute: ADD, CALCULATE, DETERMINE

  Actions: TURN ON, TURN OFF

Example:

```
:MAIN
BEGIN
Read three numbers
Calculate their sum
Display the result
END
```

IF-THEN-ELSE-ENDIF:

These keywords can be used to indicate that a decision is to be made. The general format of this construct is:

```
IF condition THEN
statement
statement
ELSE
statement
statement
ENDIF
```

Example:

```
IF temperature > 100 THEN
Turn off heater
Start the engine
ELSE
Turn on heater
ENDIF
```

REPEAT–UNTIL

This construct is used to specify a loop where the test is performed at the end of the loop. The loop continues forever if the condition is not satisfied.

The general format is:

```
REPEAT
Statement
Statement
UNTIL condition
```

Example:

```
Set counter = 0
REPEAT
Turn on LED
Wait 1 s
Turn off LED
Wait 1 s
Increment counter
UNTIL counter = 5
```

DO–WHILE

This construct is similar to REPEAT–UNTIL, but here the loop is executed while the condition is true. The condition is tested at the end of the loop. The general format is:

```
DO
statement
statement
statement
WHILE condition
```

Example:

```
Set counter = 0
DO
Turn on LED
Wait 1 s
Turn off LED
Wait 1 s
Increment counter
WHILE counter < 5
```

WHILE–WEND

This construct is similar to REPEAT–UNTIL, but here the loop may never be executed, depending on the condition. The condition is tested at the beginning of the loop.

The general format is:

```
WHILE condition
statement
statement
statement
WEND
```

Example: the loop is never executed:

```
I = 0
WHILE I > 0
Turn on LED
Wait 3 s
WEND
```

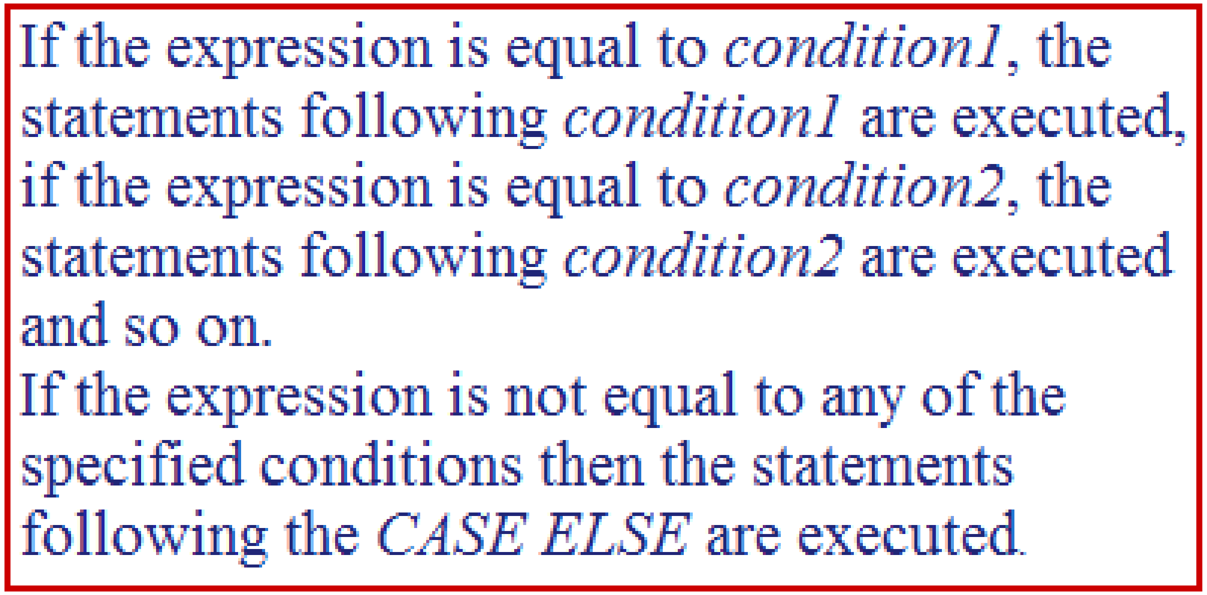
Example: the loop is executed 10 times

```
I = 0
WHILE I < 10
Turn on motor
Wait 2 seconds
Turn off motor
Increment I
WEND
```

CASE–CASE ELSE–ENDCASE

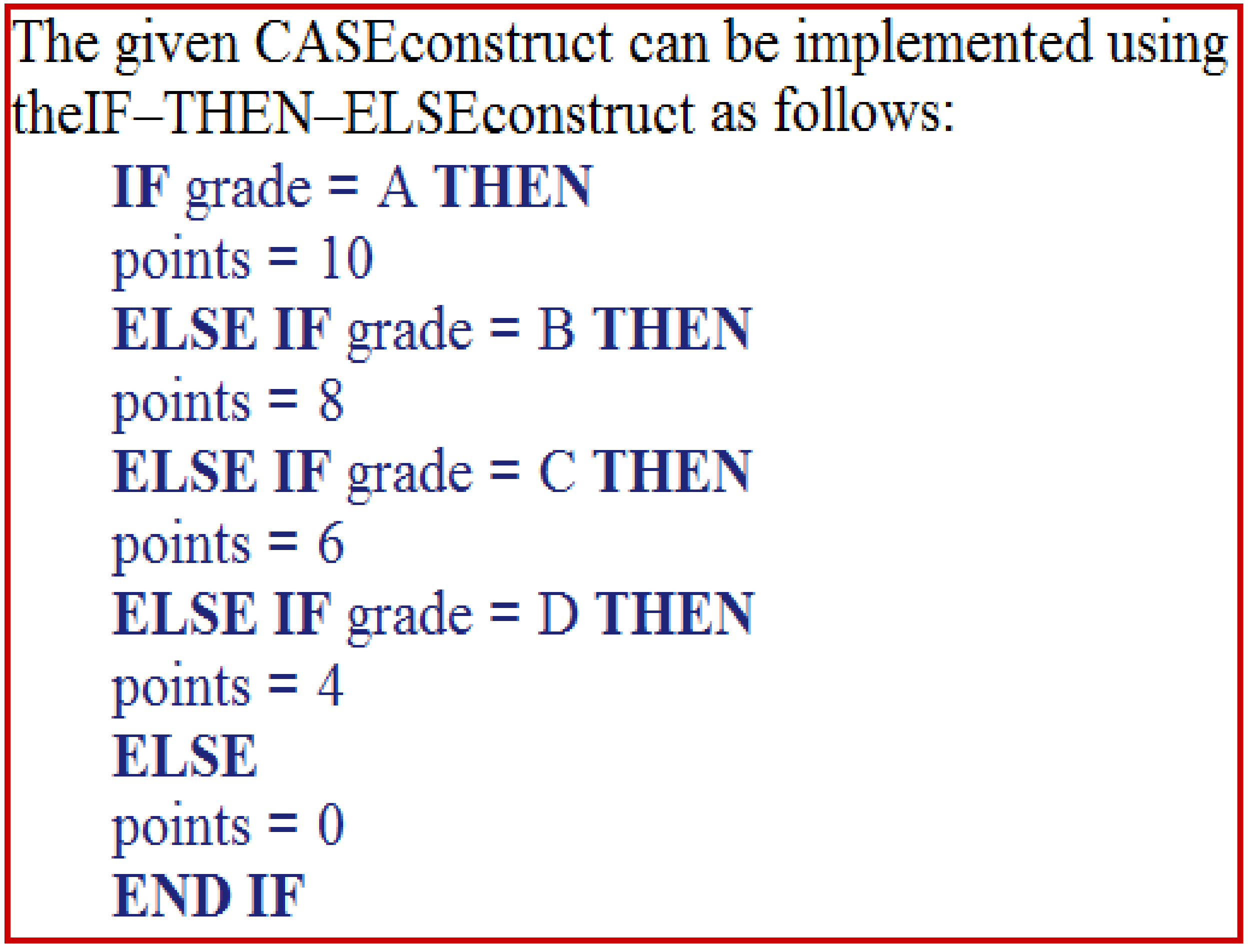
The CASE construct is used for multi-way branch operations. An expression is selected and, based on the value of this expression, a number of mutually exclusive tests can be done and statements can be executed for each case. The general format of this construct is:

```
CASE expression OF
condition1:
statement
statement
condition2:
statement
Statement
. . .
CASE ELSE
Statement
Statement
END CASE
```


If the expression is equal to *condition1*, the statements following *condition1* are executed, if the expression is equal to *condition2*, the statements following *condition2* are executed and so on.
If the expression is not equal to any of the specified conditions then the statements following the *CASE ELSE* are executed.

Example:

```
CASE grade OF
A: points = 10
B: points = 8
C: points = 6
D: points = 4
CASE ELSE points = 0
END CASE
```



The given CASEconstruct can be implemented using theIF–THEN–ELSEconstruct as follows:

```
IF grade = A THEN
points = 10
ELSE IF grade = B THEN
points = 8
ELSE IF grade = C THEN
points = 6
ELSE IF grade = D THEN
points = 4
ELSE
points = 0
END IF
```

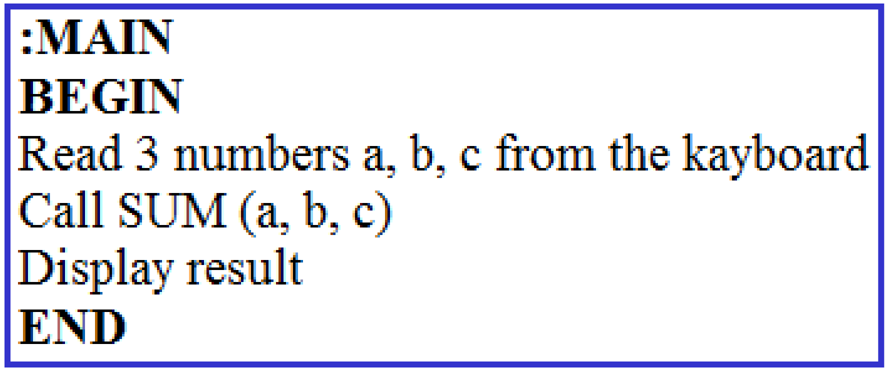
Calling Modules:

Modules can be called using the CALL keyword and then specifying the name of the module.

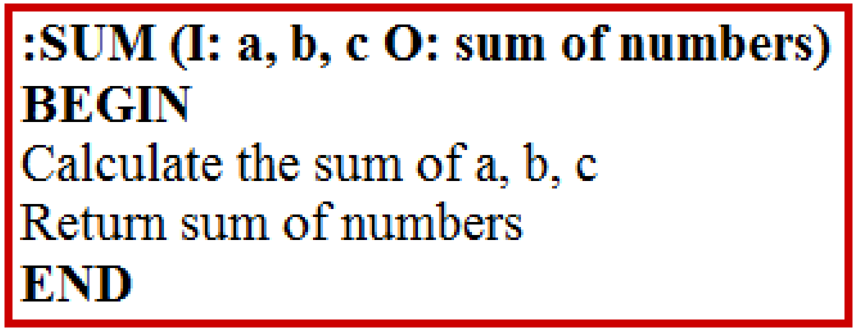
It is useful if the input parameters to be passed to the module are specified when a module is called. Similarly, at the header of the module description the input and the output parameters of a module should be specified. An example is given below.

Example:

Write the pseudocode for an application where three numbers are read from the keyboard into a main program, their sum calculated using a module called SUM, and the result displayed by the main program.



```
:MAIN
BEGIN
Read 3 numbers a, b, c from the kayboard
Call SUM (a, b, c)
Display result
END
```



```
:SUM (I: a, b, c O: sum of numbers)
BEGIN
Calculate the sum of a, b, c
Return sum of numbers
END
```

Synchronization Methods:

- One of the important features of real-time algorithms is that once they have been started they run continuously until some event occurs to stop them or until they are stopped manually by an operator.
- It is important to make sure that the loop is run continuously and exactly at the same times. This is called synchronization and there are several ways in which synchronization can be achieved in practice, such as:
  - using polling in the control algorithm;
  - using external interrupts for timing;
  - using timer interrupts;
  - ballast coding in the control algorithm;
  - using an external real-time clock.

Synchronization Methods:

1. Using Polling:

- Polling is the software technique where we keep waiting until a certain event occurs, and only then perform the required actions. This way, we wait for the next sampling time to occur and only then run the controller algorithm.
- The polling technique is used in DDC applications since the controller cannot do any other operation during the waiting of the next sampling time.
- The polling technique is described below as a sequence of steps:

```
Repeat Forever
While Not sampling time
Wait
End
 Read the desired value, R
 Read the actual plant output, Y
 Calculate the error signal, $E = R - Y$
 Calculate the controller output, U
 Send the controller output to D/A converter
End
```

2. Using External Interrupts for Timing:

- The controller synchronization task can easily be performed using an external interrupt.
- The controller algorithm can be written as an interrupt service routine (ISR).
- The external interrupt will typically be a clock with a period equal to the required sampling time. Thus, the computer will run the ISR at every sampling instant.
- At the end of the ISR control is returned to the main program where the program either waits for the occurrence of the next interrupt or can perform other tasks (e.g. displaying data on a LCD) until the next external interrupt occurs.
- The external interrupt approach provides accurate implementation of the control algorithm as far as the sampling time is concerned.
- One drawback of this method is that an external clock is required to generate the interrupt pulses.
- The external interrupt technique has the advantage that the controller is not waiting and can perform other tasks in between the sampling instants.
- The external interrupt technique of synchronization is described below as a sequence of steps:

```
Main program:
Wait for an external interrupt (or perform some other tasks)
End
```

- **Interrupt service routine (ISR):**

```
    Read the desired value, R
    Read the actual plant output, Y
    Calculate the error signal, $E = R - Y$
    Calculate the controller output, U
    Send the controller output to D/A converter
    Return from interrupt
```

3. Using Timer Interrupts:

- Another popular way to perform controller synchronization is to use the timer interrupt available on most microcontrollers.
- The controller algorithm is written inside the timer ISR, and the timer is programmed to generate interrupts at regular intervals, equal to the sampling time.
- At the end of the algorithm control returns to the main program, which either waits for the occurrence of the next interrupt or performs other tasks (e.g. displaying data on an LCD) until the next interrupt occurs.
- The timer interrupt approach provides accurate control of the sampling time. Another advantage of this technique is that no external hardware is required since the interrupts are generated by the internal timer of the microcontroller.
- The timer interrupt technique of synchronization is described as a sequence of steps:

```
Main program:
Wait for a timer interrupt (or perform some other tasks)
End
```

Interrupt service routine (ISR):

```
Read the desired value, R
Read the actual plant output, Y
Calculate the error signal, $E = R - Y$
Calculate the controller output, U
Send the controller output to D/A converter
Return from interrupt
```

4. Using Ballast Coding:

- In this technique the loop timing is made to be independent of any external or internal timing signals. The method involves finding the execution time of each instruction inside the loop and then adding *dummy* code to make the loop execution time equal to the required sampling time.
- This method has the advantage that no external or internal hardware is required. But one big disadvantage is that if the code inside the loop is changed, or if the CPU clock rate of the MC is changed, then it will be necessary to readjust the execution timing of the loop.
- The ballast coding technique of synchronization is described below as a sequence of steps. Here, it is assumed that the loop timing needs to be increased and some dummy code is added to the end of the loop to make the loop timing equal to the sampling time:

```
Do Forever:
  Read the desired value, $R$
  Read the actual plant output, $Y$
  Calculate the error signal, $E = R - Y$
  Calculate the controller output, $U$
  Send the controller output to D/A converter
Add dummy code
. . .
. . .
Add dummy code
End
```

5. Using an External Real-Time Clock:

- Some RT clock hardware is attached to the MC where the clock is updated at every *tick.* The RT clock is then read continuously and checked against the time for the next sample. Immediately on exiting from the wait loop the current value of the time is stored and then the time for the next sample is updated by adding the stored time to the sampling interval. Thus, the interval between the successive runs of the loop is independent of the execution time of the loop. Although the external clock technique gives accurate timing, it has the disadvantage that RT clock hardware is needed.
- The external RT clock technique of synchronization is described below as a sequence of steps. T is the required sampling time in ticks, which is set to n at the beginning of the algorithm. For example, if the clock rate is 50 Ticks per second, then a Tick is equivalent to 20 ms, and if the required sampling time is 100 ms, we should set $T = 5$:

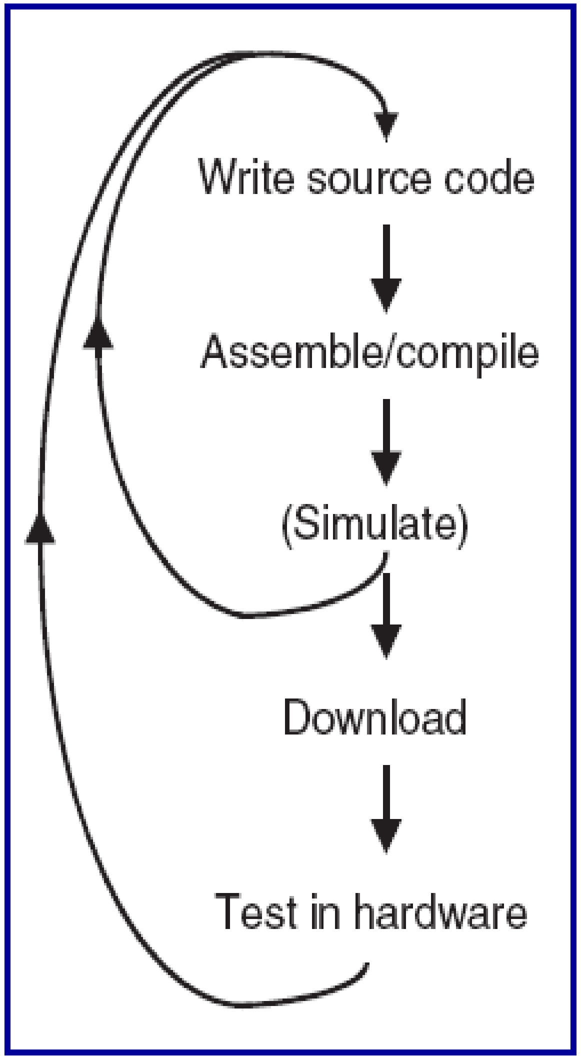
```
T = n
Next Sample Time = Ticks + T
Do Forever:
While Ticks < Next Sample Time
Wait
End
Current Time = Ticks
 Read the desired value, R
 Read the actual plant output, Y
 Calculate the error signal, E = R − Y
 Calculate the controller output, U
Send the controller output to D/A converter
 Next Sample Time=Current Time + T
End
```

Software Requirements for DERT Systems:

- Computer hardware is nowadays very fast, and control computers are generally programmed using a high-level language. The use of the assembly language is reserved for very special and time-critical applications, such as fast, real-time device drivers.
- C is a popular language used in most computer control applications. It is a powerful language that enables the programmer to perform low-level operations, without the need to use the assembly language.
- The software requirements in real-time systems can be summarized as follows:
  - the ability to read data from input ports;
  - the ability to send data to output ports;
  - internal data transfer and mathematical operations;
  - timer interrupt facilities for timing the controller algorithm.
- All of these requirements can be met by most digital computers, and, as a result, most computers can be used as controllers in digital control systems.

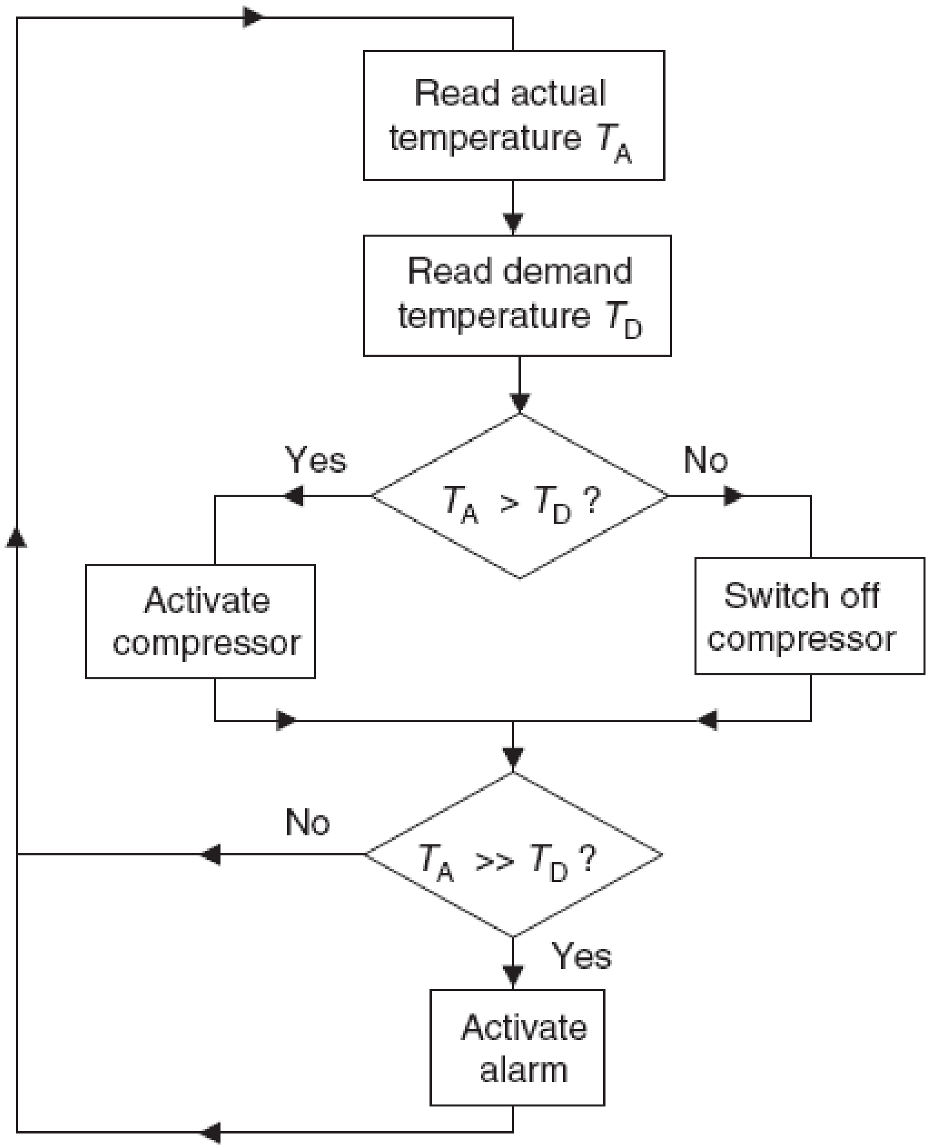
Program Development Process:

- The programmer writes the program, called the source code, in Assembler language.
- This is then assembled by the Cross-Assembler running on the host computer. The designer may choose to test the program by simulation. This is likely to lead to program.
- When satisfied with the program, the developer will then download it to the program memory of the microcontroller itself, using either a stand-alone ‘programmer’ linked to the host computer or a programming facility designed into the embedded system itself.
- The designer will then test the program running in the actual hardware. Again, this may lead to changes being required in the source code.
- To develop a simple project, a selection of different software tools is beneficial. These are usually bundled together into what is called an Integrated Development Environment (IDE), such as PROTUS and MPLAB.



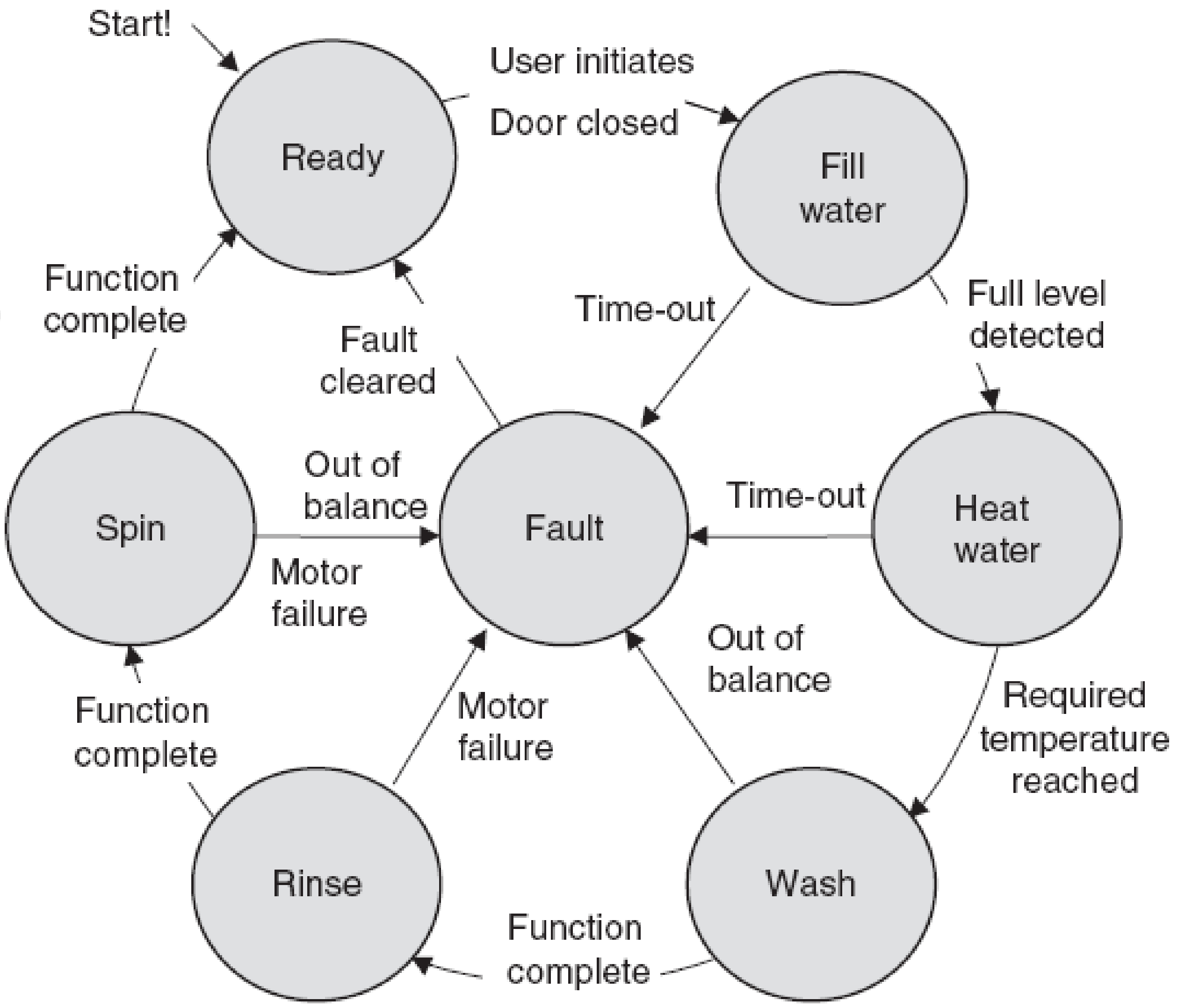
Example:

Software design using flowcharts.



Example:

Software design using State diagrams.



Assignment:

1. What are the differences between a flow chart and a structure chart?
2. What are the three major components of a structure chart? Explain the function of each component with an example.
3. Draw a flow chart to show how a fixed-time delay can be generated?
4. What are the advantages of pseudocode?
5. What are the basic components of pseudocode?
6. Write pseudocode to read the base and the height of a triangle from the keyboard, call a module to calculate the area of the triangle and display the area in the main program.
7. Explain how iteration can be done in pseudocode. Give an example?
8. Give an example of pseudocode to show how multi-way selection can be done using the CASE construct? Write the equivalent IF–ELSE–ENDIF construct?

For more information:

1. Dogan Ibrahim, “Microcontroller Based Applied Digital Control”, J.Wiley, England, 2006. ISBN 0-470-86335-8.
2. http://www.cs.ou.edu/~fagg/umass/classes/377f02/lectures.html
3. http://www.c.s.umbc.edu/~younis/Real-Time/CMSC691S.htm#D