# Software Requirements for Computer Control

Presented by:

Dr. Walid Ghoneim

Reference:

Microcontroller Based Applied Digital Control, Dogan Ibrahim

# Microcontrollers: Programming Languages

- Microcontrollers 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. !!!!!!
- The use of the assembly language is reserved for very special and time-critical applications, such as fast, real-time device drivers.
- Nowadays, microcontrollers can be programmed using high-level languages such as BASIC, PASCAL or C.

# Microcontrollers: Programming Languages

- High-level languages offer several advantages over the assembly language:
  - Easier to use in developing projects/programs.
  - Program maintenance and trouble-shooting is much easier.
  - 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.

- But high-level languages have some disadvantages as well. For example:
  - The length of the code in memory is usually larger.
  - The programs developed using the assembly language usually run faster than those developed using a high-level language.

# Microcontrollers: Programming Languages

- Best known Choice is C language, Why ?
- 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.
- And you can program with assembly language and use its instructions in the Project Developer Software and its C compiler .

# The software requirements in a control computer

- The software requirements in a control computer 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.
- Thus, most computers can be used as controllers in digital control systems.

# The software requirements in a control computer

- The software requirements in a control computer 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.
- Thus, most computers can be used as controllers in digital control systems.

- Remember: It is neither justified nor cost-effective to use a minicomputer to control the speed of a motor.
- A microcontroller is much more suitable for this kind of control application.
- But, if there is a MIMO system, or it is required to provide sophisticated control tasks or supervisory tasks, then the use of a minicomputer is justified.

# The Controller Algorithm

- The controller algorithm in a computer is implemented as a program which runs continuously in a loop which is executed at the start of every sampling time.
- Inside the loop, the desired reference value is read, the actual plant output is also read, and the difference between the desired value and the actual value is calculated, which forms the error signal.
- The control algorithm is then implemented and the controller output for this sampling instant is calculated.
- This output is sent to a D/A converter which generates an analog equivalent of the desired control action.
- This signal is then fed to an actuator/control element, which in turn drives the plant to the desired point.

# The Controller Algorithm

- The operation of the controller algorithm, assuming that the reference input and the plant output are digital signals, is summarized below as a sequence of simple steps:

  - **Repeat Forever**
    - When it is time for next sampling instant:
    - 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
    - Wait for the next sampling instant
  - **End**

# The Controller Algorithm

- Similarly, if the reference input and the plant output are analog signals, the operation of the controller algorithm can be summarized as:

  - **Repeat Forever**
    - When it is time for next sampling instant:
    - Initiate an A/D conversion for the R signal
    - Wait the till conversion ends
    - Read the desired value, *R, in digital form from the A/D converter*
    - Initiate an A/D conversion for the Y signal
    - Wait till the conversion ends
    - Read the actual plant output, *Y , in digital form from the A/D converter*
    - Calculate the error signal, $E = R - Y$
    - Calculate the controller output, $U$
    - Send the controller output to D/A converter
    - Wait for the next sampling instant
  - **End**

# Synchronization

- It is important to make sure that the loop runs continuously and exactly at the same time intervals, i.e. exactly at the sampling instants (Constant Ts).
- 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: Polling

- Polling is a software technique where we keep waiting until a certain event occurs, and only then perform the required actions.
- It refers to actively sampling the status of a device.
- This way, we wait for the next sampling time to occur and only then run the controller algorithm.
- Example: when an I/O or an ADC operation is required the computer does nothing other than check the status of the I/O or ADC device until it is ready, at which point the device is accessed.
- The polling technique is used in DDC applications since the controller cannot do any other operation during the waiting of the next sampling time.

# Synchronization: Polling

- The polling technique is described in this sequence of steps:
- **Repeat Forever**
- **While Not sampling time:**
  - Wait
- **End**
- **When Sampling Instant Occurs:**
  - 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**

# Synchronization: Using External Interrupts for Timing

- Here, the controller algorithm is written as an interrupt service routine (ISR) associated with an external interrupt.
- The external interrupt source is a clock with a period equal to the required sampling time.
- Thus, the computer will run the interrupt service routine (i.e. the controller algorithm) at every sampling instant.
- At the end of the ISR, control is returned to the main program where it either waits for the occurrence of the next interrupt or performs other tasks (e.g. displaying data on a LCD) until the next external interrupt occurs.

# Synchronization: Using External Interrupts for Timing

- 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.

# Synchronization: Using External Interrupts for Timing

- 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**

# Synchronization: Using Timer Interrupts

- Timers are available on most microcontrollers and DSPs.
- Here, the controller algorithm is written inside the timer interrupt service routine, 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.

# Synchronization: Using Timer Interrupts

- The timer interrupt technique of synchronization is described below 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**

# Synchronization: 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 microcontroller is changed, then it will be necessary to readjust the execution timing of the loop.

# Synchronization: Ballast Coding

- The steps of the ballast coding technique are described below.
- Assume that the loop timing needs to be increased, thus dummy code is added before the end 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**

# Synchronization: Using an External Real-Time Clock

- Similar to using an external interrupt for synchronization.
- Here, some real-time clock hardware is attached to the microcontroller where the clock is updated at every *tick*; for example, depending on the clock used, 50 ticks will be equal to 1 s if the tick rate is 20 ms.
- The real-time 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 real-time clock hardware is needed.

# Synchronization: Using an External Real-Time Clock

- The external real-time 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 $T = 5$.

# Synchronization: Using an External Real-Time Clock

- **Initialization:**
  - $T = n$
  - Next Sample Time = Ticks + $T$
- **Do Forever:**
  - **Read** Ticks
  - **While** Ticks < Next Sample Time
    - Wait
  - **End**
  - **Else**
    - 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**

# Good Luck