Introduction
Chapter 1
Introduction

1.0 Overview

Data acquisition and control systems need to get real-world signals into the computer. These signals come from a diverse range of instruments and sensors.

A Data Acquisition System is comprised of three parts; an I/O sub-system, a host computer and the controlling software.

There are different types of data acquisition systems they are

Serial Communication Data Acquisition Systems

Serial communication data acquisition systems are a good choice when the measurement needs to be made at a location which is distant from the computer. There are several different communication standards, RS232 is the most common but only supports transmission distances up to 50 feet. RS485 is superior to RS485 and supports transmission distances to 5,000 feet.

Parallel Port Data Acquisition Systems

The standard parallel port on a computer which is commonly used for a printer connection can also be used to connect to a data acquisition device. Parallel port systems often support very high sample rates, although the distance between the computer and the data acquisition device is limited to a few feet.

Data Acquisition Plug-In Boards
Computer data acquisition boards plug directly into the computer bus. Advantages of using boards are speed (because they are connected directly to the bus) and cost (because the overhead of packaging and power is provided by the computer). Boards offered are primarily for IBM PC and compatible computers. Features provided by the cards can vary due to number and type of inputs (voltage, thermocouple, on/off), outputs, speed and other functions provided. Each board installed in the computer is addressed at a unique Input/Output map location. The I/O map in the computer provides the address locations the processor uses to gain access to the specific device as required by its program.

**USB Data Acquisition Systems**

The Universal Serial Bus (USB) is a new standard for connecting PCs to peripheral devices such as printers, monitors, modems and data acquisition devices. USB offers several advantages over conventional serial and parallel connections, including higher bandwidth (up to 12 Mbits/s) and the ability to provide power to the peripheral device. USB is ideal for data acquisition applications. Since USB connections supply power, only one cable is required to link the data acquisition device to the PC, which most likely has at least one USB port.

![Simplified Block diagram of USB Based Data Acquisition](image)

Simplified Block diagram of USB Based Data Acquisition
Use of USB for Data Acquisition

The USB is extremely convenient for data acquisition for several reasons.

1. The equipment can obtain power from the USB; it doesn’t need to be battery powered or plugged into the wall. This makes USB ideal for portable data acquisition with a laptop.
2. Using a USB hub you can connect many devices to one socket - letting you easily expand your system should requirements grow.
3. USB ports are provided on most new PCs - no need to open the computer and install adaptor cards.
4. You can plug in and unplug your equipment without switching off your computer or even restarting Windows.
5. The USB cable can be up to 5 m long. However, using USB hubs between cables you can reach 30m.
6. Faster speeds than those allowed by RS232 connections are achievable.
7. You can use USB devices alongside existing data acquisition equipment (such as instruments that plug directly into the RS232 port).

With USB based data acquisition the common measurements include the following.

- Temperature
- Strain, tension, compression, shear force, torsional force, pressure, weight
- Displacement
- Speed (linear and rotational)
- Vibration and acoustics
- Humidity
- Liquid level
- True/false states
A transducer is required to translate the physical phenomenon into an electrical signal, and in many cases, signal conditioning is required to convert the transducers' output into a nice healthy voltage suitable for a data acquisition system's input.

How to Use Your PC to Measure Temperature

Temperature monitoring is central to the majority of data acquisition systems, be it to save energy costs, increase safety, reduce testing time...whatever its reasons, it will need a device to measure the temperature - a temperature sensor.

With the USB based Data acquisition, you can take measurements from four sensor categories:

- Thermocouple – types J, K, R, S, T, N, E, and B
- Resistance temperature detectors (RTDs) – 2, 3, or 4-wire measurements of 100 Ω platinum RTDs
- Thermistors – 2, 3, or 4-wire measurements
- Semiconductor temperature sensors – LM35 or equivalent

Semiconductor sensors are suitable over a range of approximately -40 °C to 125 °C, where an accuracy of ±2 °C is adequate. The temperature measurement range of a semiconductor sensor is small when compared to thermocouples and RTDs. However, semiconductor sensors can be accurate, inexpensive and easy to interface with other electronics for display and control. Here it uses LM35 Whose Linear output: 10 mV/°C
- -55 – 150 °C range
- 4 – 30 V input needed
- Accurate to at least ±0.75 °C
1.1 An Introduction To USB

Today, an increasing number of mobile consumer electronics products portable digital assistants (PDAs), mobile phones, digital cameras, portable storage devices, etc. use the USB interface to exchange data with host PCs. USB is becoming the preferred means of connecting devices to PCs. Whilst from the user's viewpoint, USB devices are very easy to install and use, for the developer, it can pose some serious technical challenges, particularly in the control of the USB port from Windows.

The Universal Serial Bus (USB) is a fast and flexible interface for connecting devices or peripherals to computers. The external connectivity of Input/Output devices to PC in mid-1990 was based on availability of serial ports and parallel ports. These interfaces used different connectors and cables and they offered point-to-point connections to peripherals. The serial and parallel ports could not provide capabilities to attach new peripherals to PC. By unifying various connectors, protocols and cables into one well-defined interface, a new technology called Universal Serial Bus (USB) was introduced in 1996.

Popular in the desktop PC market for several years now but now moving into the embedded field, USB (universal serial bus) is the serial bus which can realize the
Plug & Play connection for PC peripherals. In the mid-1990s, a group of engineers from the companies Compaq, DEC, IBM, Intel, Microsoft, NEC and Northern Telecom worked out the specification for a very high speed serial interface that would largely replace conventional RS232 and Centronics ports on the PC. Today, more than 1000 companies develop products which can be connected to the PC via USB. These can range from standard PC peripherals like printers and modems but now increasingly to specialist devices like scientific instruments, machine controls and even on-board networks in (racing) cars!

1.1.1 USB Mean

The abbreviation stands for "universal serial bus" - this means the specification of a serial bus for easy connection of the peripherals to the PC. It removes the need to open up a PC when adding a new peripheral device and via the PC Plug-and-Play BIOS and chipsets, allow the required software to be installed automatically. Up to 127 devices can be connected to the PC via this bus.

1.1.2 Connection of Devices

The first and commonest USB PC peripherals are the standard human interface devices such as the keyboard, mouse and joystick. Latterly, "high speed" units like ISDN modems, scanners, printers, external hard disks and CD ROM drives have appeared. The high transmission rate of 12 Mbit/s permits even MPEG-2 based video products to be connected. With the appearance of low-cost USB-equipped microcontrollers from Intel, Infineon, ST, Cypress and others, the manufacturer of specialist products can embedded a USB "endpoint" in their PC-hosted product and gain all the advantages of easy installation and high communication speed.

1.1.3 Bus Connection
In order to realize USB's advantages there are certain standards which allow the interoperability of all components. Firstly there is a mechanical standard for the plug and the socket which physically create the basis for USB connection. Then you need to have a basic understanding of the following terms:

**Host:** This is the device which controls the bus, and the data transmission. Generally, this is a PC. USB-capable PCs for example offer a PCI chipset in which the USB host controller is already integrated. The bulk of the host controller software is part of the Windows operating system. However, unless your intended USB product is a standard human interface device like a keyboard or a mouse, Windows will not know how to handle it. Thus you will have to provide some custom software running under the Windows operating system - addressing the USB port on the PC is something that Windows does not make easy but fortunately there is a solution!

**Function:** This is the peripheral device which is connected to the host over the bus, and provides input/output information via the bus to the host. Again you need a suitable controller which handles who sends what on the bus, and the data exchange.

**Hub:** This is the communication distribution centre which allows the connection of further USB peripherals to the bus. The most important devices such as monitor or keyboard have this connection option for further peripherals. Technically it can be integrated also into any other device. This way you can create tree-shaped net structures, the individual connection should not exceed 5 meters, however, max. Limit of devices to be connected is 127. Intel offers controllers in the family 8x930Hx family which can be used as communication distributors. This controller includes the "function" option.
1.1.4 USB Advantages

1. Many low-cost, standardized devices are available.
2. USB devices are compliance-tested and certified, ensuring uniform performance.
3. Using hubs, many USB devices can share one connector.
4. USB devices can tell you about their capabilities.
5. USB provides device power.
6. USB supports hot plugging. (i.e. a device can be connected or disconnected even when the system is ON)
7. USB devices can be put to sleep, be awakened, or provide a system wakeup call.
8. USB protocol takes care of low-level details like error checking and flow control.
9. The storage space for a design can be dynamically added or removed at any time.

1.1.5 USB Versions:
4. The OTG-supplement to the USB 2.0 was approved in December 2001.

1.1.8 USB Speeds:

1. USB operates at three speeds
2. Low speed, 1.5 Mbit/sec
3. Full speed, 12 Mbit/sec
4. High speed (USB 2.0), 480 Mbit/sec

1.1.7 Speed Identification:

1. USB devices indicate their speed by pulling either the D+ or D- Line to 3.3 Volts.
2. Device connected to host having HIGH on D- line is low speed device.
3. Device connected to host having HIGH on D+ line is full/high speed device.
4. Hosts and hubs support all 3 speeds.

![Diagram of USB Speeds and Speed Identification](image)

Dual-role device with Mini-AB connector

Default Device A

Dual-role device with Mini-AB connector

Default Device B

Figure 1.3 Default Bias Resistor Connections
1.1.8 Cables:

USB uses shielded cables with four wires: Vbus (4.4-5.25 volts), GND, Data (+) and Data (-). USB uses half-duplex differential signaling on its D+ and D-lines. Every USB device contains a Serial Interface Engine (SIE) that handles low level signaling details such as error checking, bit-stuffing and serializing/deserializing the data. The SIE delivers and consumes data bytes and does the wire-level signaling.

![USB Cable Diagram](image)

Figure 1.4 USB Cable

1.1.9 Topology:

We can connect many devices to USB host up to max of 127 devices using hubs. These hubs can act as either hosts or devices and they consume power from host or from external power supply. This hub contains addresses of devices connected to them. The connection topology looks as follow.

![Topology Diagram](image)

Figure 1.5 A Tiered Star "P" Indicates A Peripheral Device
USB uses a tiered star topology. The host is the sole master. Only the host can initiate USB transfers. USB is not a peer-to-peer bus, so peripherals don't talk to each other. The host sends requests downstream through all hubs (tiers), to all the connected devices (each hub is the center of a star). Every host request includes a device address and an endpoint number (an endpoint is an addressable FIFO inside a device), so that only one device, the addressed one, responds by sending data upstream.

1.1.10 Hubs:

USB hubs provide a means to add devices to a USB system. A USB system can have up to five attached hubs; the limit is due to hub and cable propagation times. The Vbus spec of 4.4 volts minimum accounts for the maximum voltage drop through five series-connected external hubs.

Every PC contains a root hub in the host controller. The root hub provides 500 milliamps of Vbus power per connector. An external hub can be bus powered, drawing its power from the Vbus wire, if (a) the hub itself consumes 100 milliamps or less, and (b) it has four or fewer downstream ports that supply 100 milliamps or less. This keeps the bus power consumption within the host's 500 milliamp limit.

1.1.11 Endpoints:

The host broadcasts requests downstream, using a preamble that contains a device address, a 4-bit endpoint number, and a direction bit. This allows any device to have up to 32 endpoints, or channels. One endpoint (Endpoint 0) is mandatory in every device for host control, but the others can be assigned any way the designer wishes. For example, endpoint 1-OUT may be output data, endpoint 1-IN may input data, endpoint 2-IN may be status information, and endpoint 3-OUT
may be configuration information. Endpoint numbers are arbitrary, assigned by the designer and reported to the host during enumeration.

1.1.12 Enumeration:

Enumeration is the process of the initial exchange of information by the host to know about the device and assign a device driver.

Steps:
1. The user plugs a device into a USB port.
2. The hub/host detects the device.
3. The host learns about the device.
4. The hub/host detects the speed of the device.
5. The hub/host resets the device.
6. The hub establishes a signal path between the device and the hub/host.
7. The host assigns an address.
8. The host learns about the device's ability.
9. The host assigns and loads a device driver.
   The host's device driver selects a configuration.

1.1.13 USB Transfers:

1.1.13.1 Data Transfer Types:

USB defines four transfer types to handle different delivery requirements. Every endpoint is declared to be of one of these transfer types, and it handles only that transfer type. The host learns about all of a device's endpoints and their transfer types during enumeration. USB transfers data in 1-millisecond frames. A special marker, called a Start Of Frame (SOF) token, delineates frames and supplies an incrementing 11-bit frame number.
CONTROL Transfers: control transfers configure and operate devices. Being “mission critical”, they employ the highest level of error checking. They are the only transfer type to have defined structure in the USB Specification. CONTROL transfers consist of up to three stages:

1. The SETUP stage contains the 8-byte request
2. The optional DATA stage contains data, if required.
3. The STATUS stage provides an additional handshake.

Control transfers are used for command and status operations. It guarantees error free data delivery (CRC, handshake).

BULK transfers: Bulk transfers use error detection and transmission re-tries (handshakes), and are scheduled on an “as available” bandwidth basis. BULK transfers are for data that requires guaranteed accuracy, but not guaranteed delivery time. Bulk transfers contain data and status stage.

INTERRUPT transfers: Interrupt transfers are a special case of BULK transfers. INTERRUPT transfers are designed for devices that supply relatively small amounts of data at unexpected times, such as keyboards and mice.

ISOCHRONOUS transfers: Isochronous transfers guarantee delivery time, but not accuracy. They are designed for high bandwidth.

USB two signal lines carry data to and from all of the devices on the bus. The wires form a single transmission path that all of the devices must share. USB’s pair of wires carries a single differential signal. It manages the traffic by dividing the time into frames or micro frames. The host gives each transfer a portion of each frame or micro frame. For low and full speed data, frames are divided into millisecond and for high-speed data the host divides each frame into eight 125-microsecond frames.
Each transfer in USB consists of one or more transactions, and each transaction in turn consists of one, two or three packets.

![Diagram of USB Transfer]

Each transaction has up to three phases or parts that occur in sequence: token data and handshake as shown in the below figure (Figure 2.4). Each phase consists of one or two transmitted packets. Each packet is a block of information with a defined format. All packets begin with a Packet ID (PID) that contains identifying information.

1.1.13.2 Handshakes:

An ACK handshake means, "I got the data without errors". A NAK handshake means, "I got the request without errors, but I'm busy, try again later". A bus error is indicated by no response.

1.1.14 Power Management
USB power consumption is carefully controlled. When a USB device is first attached to the bus, the host queries the device as part of a process called enumeration. One of the first things the host asks a device is how much power it requires. When a device first connects, it must consume 100 milliamps or less to keep it within the bus-powered hub limit. If the host discovers during enumeration that a device requires more than 100 milliamps once it's operating ("configured", in USB-speak), and it's plugged into a bus-powered hub, the host refuses to let the device on the bus simply by not configuring it.

USB also has a standard way of putting a peripheral into a very low power mode (Suspend), and waking it up (Resume). There's also a mechanism by which a suspended peripheral device can initiate a Resume sequence using a method called Remote Wakeup. A suspended USB device cannot draw more than 500 micro amps from Vbus.

1.1.15 USB On The Go (OTG) protocol:

A new USB Working Group specification called "On-The-Go" (OTG) introduced in version 2.0. The OTG introduces the Dual-Role Device (DRD), which can act either as a host or a peripheral. When digital camera is connected to a printer (which are supporting USB functionality), then camera should act as host and printer as a device. In certain cases device wants send a request to the host and to implement this concept a Session Request Protocol (SRP) is defined in a way that allows device to request to turn on Vbus to initiate the session.
In the case of connecting two digital cameras, one should act as host and other as device. The A-end of the cable establishes the default-host and B-end establishes default-device. User does not know how to connect and which connector to which device. In order to avoid this confusion a method called Host Negotiation Protocol (HNP) has been introduced to switch the properties of host and device by driving Vbus and pull-down resistor.

1.1.16 USB Device Descriptors:
Descriptors are the data structures, or formatted blocks of information, that enable the host to learn about the device or initiate the device. Each descriptor contains information about either the device as a whole or an element in the device. Any USB version must support the existing basic standard descriptors. Apart from the basic descriptor, some chips contain vendor specific descriptors and other descriptors.

1.1.17 USB standard descriptors

1. Device Descriptor
2. Device qualifier
3. Configuration Descriptors
4. Other-speed configuration Descriptor
5. Interface Descriptors
6. Endpoint Descriptors
7. String Descriptors
**Device Descriptor:**

The device descriptor of a USB device represents the entire device. As a result, a USB device can only have one device descriptor. It specifies some basic, yet important information about the device such as the supported USB version, maximum packet size, vendor and product IDs and the number of possible configurations the device can have. The format of the device descriptor is shown below.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Size</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>bLength</td>
<td>1</td>
<td>Number</td>
<td>Size of the Descriptor in Bytes (18 bytes)</td>
</tr>
<tr>
<td>1</td>
<td>bDescriptorType</td>
<td>1</td>
<td>Constant</td>
<td>Device Descriptor (0x01)</td>
</tr>
<tr>
<td>2</td>
<td>bcdUSB</td>
<td>2</td>
<td>BCD</td>
<td>USB Specification Number which device complies too.</td>
</tr>
<tr>
<td>4</td>
<td>bDeviceClass</td>
<td>1</td>
<td>Class</td>
<td>Class Code (Assigned by USB Org)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If equal to Zero, each interface specifies it's own class code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If equal to 0xFF, the class code is vendor specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Otherwise field is valid Class Code.</td>
</tr>
<tr>
<td>5</td>
<td>bDeviceSubClass</td>
<td>1</td>
<td>SubClass</td>
<td>Subclass Code (Assigned by</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>bMaxPacketSize</td>
<td>1 Number</td>
<td>Maximum Packet Size for Zero Endpoint. Valid Sizes are 8, 16, 32, 64</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>idVendor</td>
<td>2 ID</td>
<td>Vendor ID (Assigned by USB Org)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>idProduct</td>
<td>2 ID</td>
<td>Product ID (Assigned by Manufacturer)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>bcdDevice</td>
<td>2 BCD</td>
<td>Device Release Number</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>iManufacturer</td>
<td>1 Index</td>
<td>Index of Manufacturer String Descriptor</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>iProduct</td>
<td>1 Index</td>
<td>Index of Product String Descriptor</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>iSerialNumber</td>
<td>1 Index</td>
<td>Index of Serial Number String Descriptor</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>bNumConfigurations</td>
<td>1 Integer</td>
<td>Number of Possible Configurations</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2.1 Device descriptor**

The value is in binary coded decimal with a format of 0xJJMN where JJ is the major version number, M is the minor version number and N is the sub minor version number. e.g. USB 2.0 is reported as 0x0200, USB 1.1 as 0x0110 and USB 1.0 as 0x0100.
1. The **bDeviceClass**, **bDeviceSubClass** and **bDeviceProtocol** are used by the operating system to find a class driver for your device. Typically only the bDeviceClass is set at the device level. Most class specifications choose to identify itself at the interface level and as a result set the bDeviceClass as 0x00. This allows for the one device to support multiple classes.

2. The **bMaxPacketSize** field reports the maximum packet size for endpoint zero. All devices must support endpoint zero.

3. The **idVendor** and **idProduct** are used by the operating system to find a driver for your device. The Vendor ID is assigned by the USB-IF.

4. The **bcdDevice** has the same format than the bcdUSB and is used to provide a device version number. The developer assigns this value.

5. Three string descriptors exist to provide details of the manufacturer, product and serial number. There is no requirement to have string descriptors. If no string descriptor is present, a index of zero should be used.

6. **bNumConfigurations** defines the number of configurations the device supports at its current speed.

### Configuration Descriptors

A USB device can have several different configurations although the majority of devices are simple and only have one. The configuration descriptor specifies how the device is powered, what the maximum power consumption is, the number of interfaces it has. Therefore it is possible to have two configurations, one for when the device is bus powered and another when it is mains powered. As this is a "header" to the Interface descriptors, its also feasible to have one configuration using a different transfer mode to that of another configuration. Once the host has examined all the configurations, the host will send a **SetConfiguration** command with a non-zero value, which matches the **bConfigurationValue** of one of the configurations. This is used to select the desired configuration.
### Configuration descriptor

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Size</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>bLength</td>
<td>1</td>
<td>Number</td>
<td>Size of Descriptor in Bytes</td>
</tr>
<tr>
<td>1</td>
<td>bDescriptorType</td>
<td>1</td>
<td>Constant</td>
<td>Configuration Descriptor (0x02)</td>
</tr>
<tr>
<td>2</td>
<td>wTotalLength</td>
<td>2</td>
<td>Number</td>
<td>Total length in bytes of data returned</td>
</tr>
<tr>
<td>4</td>
<td>bNumInterfaces</td>
<td>1</td>
<td>Number</td>
<td>Number of Interfaces</td>
</tr>
<tr>
<td>5</td>
<td>bConfigurationValue</td>
<td>1</td>
<td>Number</td>
<td>Value to use as an argument to select this configuration</td>
</tr>
<tr>
<td>6</td>
<td>iConfiguration</td>
<td>1</td>
<td>Index</td>
<td>Index of String Descriptor describing this configuration</td>
</tr>
<tr>
<td>7</td>
<td>bmAttributes</td>
<td>1</td>
<td>Bitmap</td>
<td>D7 Reserved, set to 1. (USB 1.0 Bus Powered)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D6 Self Powered</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D5 Remote Wakeup</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>D4..0 Reserved, set to 0.</td>
</tr>
<tr>
<td>8</td>
<td>bMaxPower</td>
<td>1</td>
<td>mA</td>
<td>Maximum Power Consumption in 2mA units</td>
</tr>
</tbody>
</table>
1. When the configuration descriptor is read, it returns the entire configuration hierarchy which includes all related interface and endpoint descriptors. The `wTotalLength` field reflects the number of bytes in the hierarchy.

2. `bNumInterfaces` specifies the number of interfaces present for this configuration.

3. `bConfigurationValue` is used by the SetConfiguration request to select this configuration.

4. `iConfiguration` is a index to a string descriptor describing the configuration in human readable form.

5. `bmAttributes` specify power parameters for the configuration. If a device is self powered, it sets D6. Bit D7 was used in USB 1.0 to indicate a bus powered device, but this is now done by `bMaxPower`. If a device uses any power from the bus, whether it be as a bus powered device or as a self powered device, it must report its power consumption in `bMaxPower`. Devices can also support remote wakeup which allows the device to wake up the host when the host is in suspend.

6. `bMaxPower` defines the maximum power the device will drain from the bus. This is in 2mA units, thus a maximum of approximately 500mA can be specified. The specification allows a high powered bus powered device to drain no more than 500mA from Vbus. If a device loses external power, then it must not drain more than indicated in `bMaxPower`. It should fail any operation it cannot perform without external power.

**Interface Descriptors**

The interface descriptor could be seen as a header or grouping of the endpoints into a functional group performing a single feature of the device. The interface descriptor conforms to the following format,

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Size</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>bLength</td>
<td>1</td>
<td>Number</td>
<td>Size of Descriptor in Bytes (9 Bytes)</td>
</tr>
<tr>
<td></td>
<td>bDescriptorType</td>
<td>1</td>
<td>Constant</td>
<td>Interface Descriptor (0x04)</td>
</tr>
<tr>
<td>---</td>
<td>----------------</td>
<td>---</td>
<td>----------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>2</td>
<td>bInterfaceNumber</td>
<td>1</td>
<td>Number</td>
<td>Number of Interface</td>
</tr>
<tr>
<td>3</td>
<td>bAlternateSetting</td>
<td>1</td>
<td>Number</td>
<td>Value used to select alternative setting</td>
</tr>
<tr>
<td>4</td>
<td>bNumEndpoints</td>
<td>1</td>
<td>Number</td>
<td>Number of Endpoints used for this interface</td>
</tr>
<tr>
<td>5</td>
<td>bInterfaceClass</td>
<td>1</td>
<td>Class</td>
<td>Class Code (Assigned by USB Org)</td>
</tr>
<tr>
<td>6</td>
<td>bInterfaceSubClass</td>
<td>1</td>
<td>SubClass</td>
<td>Subclass Code (Assigned by USB Org)</td>
</tr>
<tr>
<td>7</td>
<td>bInterfaceProtocol</td>
<td>1</td>
<td>Protocol</td>
<td>Protocol Code (Assigned by USB Org)</td>
</tr>
<tr>
<td>8</td>
<td>bInterface</td>
<td>1</td>
<td>Index</td>
<td>Index of String Descriptor Describing this interface</td>
</tr>
</tbody>
</table>

**Interface descriptor**

1. **bInterfaceNumber** indicates the index of the interface descriptor. This should be zero based, and incremented once for each new interface descriptor.

2. **bAlternateSetting** can be used to specify alternative interfaces. These alternative interfaces can be selected with the Set Interface request.

3. **bNumEndpoints** indicates the number of endpoints used by the interface. This value should exclude endpoint zero and is used to indicate the number of endpoint descriptors to follow.

4. **bInterfaceClass, bInterfaceSubClass and bInterfaceProtocol** can be used to specify supported classes (e.g. HID, communications, mass storage...
etc.) This allows many devices to use class drivers preventing the need to write specific drivers for your device.

5. **interface** allows for a string description of the interface.

**Endpoint Descriptors**

Endpoint descriptors are used to describe endpoints other than endpoint zero. Endpoint zero is always assumed to be a control endpoint and is configured before any descriptors are even requested. The host will use the information returned from these descriptors to determine the bandwidth requirements of the bus.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Size</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>bLength</td>
<td>1</td>
<td>Number</td>
<td>Size of Descriptor in Bytes (7 bytes)</td>
</tr>
<tr>
<td>1</td>
<td>bDescriptorType</td>
<td>1</td>
<td>Constant</td>
<td>Endpoint Descriptor (0x05)</td>
</tr>
</tbody>
</table>
| 2      | bEndpointAddress     | 1    | Endpoint | **EndpointAddress**  
|        |                      |      |        | Bits 0..3b Endpoint Number.    |
|        |                      |      |        | Bits 4..6b Reserved. Set to Zero |
|        |                      |      |        | Bits 7 Direction 0 = Out, 1 = In |
|        |                      |      |        | (Ignored for Control Endpoints) |
| 3      | bmAttributes         | 1    | Bitmap | Bits 0..1 Transfer Type        |
|        |                      |      |        | 00 = Control                    |
|        |                      |      |        | 01 = Isochronous                |
|        |                      |      |        | 10 = Bulk                       |
|        |                      |      |        | 11 = Interrupt                  |
|        |                      |      |        | Bits 2..7 are reserved. If Isochronous endpoint, |
Bits 3..2 = Synchronization Type (Iso Mode)
00 = No Synchronization
01 = Asynchronous
10 = Adaptive
11 = Synchronous

Bits 5..4 = Usage Type (Iso Mode)
00 = Data Endpoint
01 = Feedback Endpoint
10 = Explicit Feedback Data Endpoint
11 = Reserved

<table>
<thead>
<tr>
<th></th>
<th>wMaxPacketSize</th>
<th>2</th>
<th>Number</th>
<th>Maximum Packet Size this endpoint is capable of sending or receiving</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>bInterval</th>
<th>1</th>
<th>Number</th>
<th>Interval for polling endpoint data transfers. Value in frame counts. Ignored for Bulk &amp; Control Endpoints. Isochronous must equal 1 and field may range from 1 to 255 for interrupt endpoints.</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table End point descriptor

1. **b EndpointAddress** indicates what endpoint this descriptor is describing.

2. **bmAttributes** specifies the transfer type. This can either be Control, Interrupts, Isochronous or Bulk Transfers. If an Isochronous endpoint is specified, additional attributes can be selected such as the Synchronization and usage types.
3. **wMaxPacketSize** indicates the maximum payload size for this endpoint.

4. **bInterval** is used to specify the polling interval of certain transfers. The units are expressed in frames, thus this equates to either 1ms for low/full speed devices and 125us for high-speed devices.

**String Descriptors:**

String descriptors provide human readable information and are optional. If they are not used, any string index fields of descriptors must be set to zero indicating there is no string descriptor available.

The strings are encoded in the Unicode format and products can be made to support multiple languages. String Index 0 should return a list of supported languages. A list of USB Language IDs can be found in Universal Serial Bus Language Identifiers (LANGIDs) version 1.0.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Size</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>bLength</td>
<td>1</td>
<td>Number</td>
<td>Size of Descriptor in Bytes</td>
</tr>
<tr>
<td>1</td>
<td>bDescriptorType</td>
<td>1</td>
<td>Constant</td>
<td>String Descriptor (0x03)</td>
</tr>
<tr>
<td>2</td>
<td>wLANGID[0]</td>
<td>2</td>
<td>number</td>
<td>Supported Language Code Zero (e.g. 0x0409 English - United States)</td>
</tr>
<tr>
<td>4</td>
<td>wLANGID[1]</td>
<td>2</td>
<td>number</td>
<td>Supported Language Code One (e.g. 0x0c09 English - Australian)</td>
</tr>
<tr>
<td>n</td>
<td>wLANGID[x]</td>
<td>2</td>
<td>number</td>
<td>Supported Language Code x (e.g. 0x0407 German - Standard)</td>
</tr>
</tbody>
</table>

**String descriptor**
The above String Descriptor shows the format of String Descriptor Zero. The host should read this descriptor to determine what languages are available. If a language is supported, it can then be referenced by sending the language ID in the wIndex field of a Get Descriptor(String) request.

All subsequent strings take on the format below,

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field</th>
<th>Size</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>bLength</td>
<td>1</td>
<td>Number</td>
<td>Size of Descriptor in Bytes</td>
</tr>
<tr>
<td>1</td>
<td>bDescriptorType</td>
<td>1</td>
<td>Constant</td>
<td>String Descriptor (0x03)</td>
</tr>
<tr>
<td>2</td>
<td>bString</td>
<td>n</td>
<td>Unicode</td>
<td>Unicode Encoded String</td>
</tr>
</tbody>
</table>

Table 2.6 String format

**Device Qualifier:**

Devices that support both full and high speeds must have a device qualifier descriptor. The descriptor includes information about the descriptor itself, the device, its configurations and its classes.

**Other-Speed Configuration Descriptor:**

The other descriptor unique to devices that support both full and high speeds is the other speed configuration descriptor. This descriptor has subordinate descriptors the same as the configuration descriptors.

**1.1.17 Device Classes:**

The USB world provides a formal mechanism to allow groups of companies that work with common device types (such as mass storage, audio or input-output devices) to define and standardize their own protocols that build on the USB specification. Importantly, this mechanism (administered by the USB
Implementers Forum, Inc.) provides a path that leads to ratification of these “class” specifications and eventual incorporation into mainstream operating systems.

1.2.0 A Brief Introduction of Microcontrollers

Microcontrollers, as the name suggests, are small controllers. They are like single chip computers that are often embedded into other systems to function as processing/controlling unit. For example, the remote control you are using probably has microcontrollers inside that do decoding and other controlling functions. They are also used in automobiles, washing machines, microwave ovens, toys ... etc, where automation is needed.

1.2.1 The key features of microcontrollers include:

- **High Integration of Functionality**
  Microcontrollers sometimes are called single-chip computers because they have on-chip memory and I/O circuitry and other circuitries that enable them to function as small standalone computers without other supporting circuitry.

- **Field Programmability, Flexibility**
  Microcontrollers often use EEPROM or EPROM as their storage device to allow field programmability so they are flexible to use. Once the program is tested to be correct then large quantities of microcontrollers can be programmed to be used in embedded systems.

- **Easy to Use**
  Assembly language is often used in microcontrollers and since they usually follow a RISC architecture, the instruction set is small. The development package of microcontrollers often includes an assembler, a simulator, a programmer to
"burn" the chip and a demonstration board. Some packages include a high level language compiler such as a C compiler and more sophisticated libraries.

What is a microcontroller?

Basically, a microcontroller is a device which integrates a number of the components of a microprocessor system onto a single microchip. So a microcontroller combines onto the same microchip:

1.2.2 LPC2148 microcontroller

These powerful yet cost-effective microcontrollers offer USB 2.0 full-speed (12 Mbps) capability and have up to 512 KB of ISP/IAP Flash and up to 40 KB of SRAM. Each has up to two 10-bit A/D converters, a 10-bit D/A converter, two I2C-bus interfaces, and Fast I/O.

The LPC2148 microcontroller is based on a 16-bit/32-bit ARM7TDMI-S CPU with real-time emulation and embedded trace support, that combine microcontroller with embedded high-speed flash memory ranging from 32 kB to 512 kB. A 128-bit wide memory interface and a unique accelerator architecture enable 32-bit code execution at the maximum clock rate. For critical code size applications, the alternative 16-bit Thumb mode reduces code by more than 30 % with minimal performance penalty. Due to their tiny size and low power consumption, LPC2148 is ideal for applications where miniaturization is a key requirement, such as access control and point-of-sale. Serial communications interfaces ranging from a USB 2.0 Full-speed device, multiple UARTs, SPI, SSP to I2C-bus and on-chip SRAM of 8 kB up to 40 kB, make these devices very well suited for communication gateways and protocol converters, soft modems, voice recognition and low end imaging.
providing both large buffer size and high processing power. Various 32-bit timers, single or dual 10-bit ADC(s), 10-bit DAC, PWM channels and 45 fast GPIO lines with up to nine edge or level sensitive external interrupt pins make these microcontrollers suitable for industrial control and medical systems.

1.2.3 Key features

- 16-bit/32-bit ARM7TDMI-S microcontroller in a tiny LQFP64 package.

- 8 kb to 40 kb of on-chip static RAM and 32 kb of on-chip static RAM and 512 kb of on-chip flash memory. 128-bit wide interface/accelerator enables high-speed 60 MHz operation. In-System Programming/In-Application Programming (ISP/IAP) via on-chip boot loader software. Single flash sector or full chip erase in 400 ms and programming of 256 bytes in 1 ms.

- EmbeddedICE RT and Embedded Trace interfaces offer real-time debugging with the on-chip Real Monitor software and high-speed tracing of instruction execution.

USB 2.0 Full-speed compliant device controller with 2 kB of endpoint RAM. In addition, the LPC2146/48 provides 8 kB of on-chip RAM accessible to USB by DMA.

- One or two (LPC2141/42 vs. LPC2148) 10-bit ADCs provide a total of 6/14 analog inputs, with conversion times as low as 2.44 ms per channel.

- Single 10-bit DAC provides variable analog output (LPC2148 only).

Two 32-bit timers/external event counters (with four capture and four compare channels each), PWM unit (six outputs) and watchdog.

- Low power Real-Time Clock (RTC) with independent power and 32 kHz clock input.
- Multiple serial interfaces including two UARTs (16C550), two Fast I2C-bus (400 kbit/s), SPI and SSP with buffering and variable data length capabilities.
- Vectored Interrupt Controller (VIC) with configurable priorities and vector addresses.
- Up to 45 of 5 V tolerant fast general purpose I/O pins in a tiny LQFP64 package.
- Up to 21 external interrupt pins available.
- 60 MHz maximum CPU clock available from programmable on-chip PLL with settling time of 100 ms.

On-chip integrated oscillator operates with an external crystal from 1 MHz to 25 MHz.

- Power saving modes include Idle and Power-down.
- Individual enable/disable of peripheral functions as well as peripheral clock scaling for additional power optimization.
- Processor wake-up from Power-down mode via external interrupt or BOD. Single power supply chip with POR and BOD circuits:
  - CPU operating voltage range of 3.0 V to 3.6 V (3.3 V ± 10 %) with 5 V tolerant I/O pads.

The ARM7TDMI-S is a general purpose 32-bit microprocessor, which offers high performance and very low power consumption. The ARM architecture is based on Reduced Instruction Set Computer (RISC) principles, and the instruction set and related decode mechanism are much simpler than those of microprogrammed Complex Instruction Set Computers (CISC). This simplicity results in a high instruction throughput and impressive real-time interrupt response from a small and cost-effective processor core.

Pipeline techniques are employed so that all parts of the processing and memory systems can operate continuously. Typically, while one instruction is being
executed, its successor is being decoded, and a third instruction is being fetched from memory.

The ARM7TDMI-S processor also employs a unique architectural strategy known as Thumb, which makes it ideally suited to high-volume applications with memory restrictions, or applications where code density is an issue. The key idea behind Thumb is that of a super-reduced instruction set. Essentially, the ARM7TDMI-S processor has two instruction sets:

• The standard 32-bit ARM set.
• A 16-bit Thumb set.

The Thumb set's 16-bit instruction length allows it to approach twice the density of standard ARM code while retaining most of the ARM's performance advantage over a traditional 16-bit processor using 16-bit registers. This is possible because Thumb code operates on the same 32-bit register set as ARM code. Thumb code is able to provide up to 65% of the code size of ARM, and 160% of the performance of an equivalent ARM processor connected to a 16-bit memory system.

The particular flash implementation in the LPC2148 allows for full speed execution also in ARM mode. It is recommended to program performance critical and short code sections (such as interrupt service routines and DSP algorithms) in ARM mode. The impact on the overall code size will be minimal but the speed can be increased by 30% over Thumb mode.

1.2.4. On-chip flash program memory

The LPC2148 incorporate a 512 kB flash memory system respectively. This memory may be used for both code and data storage. Programming of the flash memory may be accomplished in several ways. It may be programmed In System via the serial port. The application program may also erase and/or program the flash while the application is running, allowing a great degree of flexibility for data
storage field firmware upgrades, etc. Due to the architectural solution chosen for an on-chip boot loader, flash memory available for user's code on LPC2148 is 500 kB. The LPC2148 flash memory provides a minimum of 100,000 erase/write cycles and 20 years of data-retention.

1.2.5. On-chip static RAM

On-chip static RAM may be used for code and/or data storage. The SRAM may be accessed as 8-bit, 16-bit, and 32-bit. The LPC2148 provide 32 kB of static RAM. In this an 8 kB SRAM block intended to be utilized mainly by the USB can also be used as a general purpose RAM for data storage and code storage and execution.

1.2.6. Memory map

The LPC2148 memory map incorporates several distinct regions. In addition, the CPU interrupt vectors may be remapped to allow them to reside in either flash memory (the default) or on-chip static RAM.

1.3 Introduction Sensors

Sensors are essential components in many applications, not only in the industries for process control but also in daily life for buildings safety and security monitoring, traffic flow measuring, weather condition monitoring and etc. In Temperature monitoring and need to be measured, thus sensors have always been given the task for doing so.

Qualitatively, the temperature of an object determines the sensation of warmth or coldness felt by touching it. More specifically, temperature is a measure of the average kinetic energy of the particles in a sample of matter, expressed in units of degrees on a standard scale.
1.3.1. RTDs

Resistance temperature detectors (RTDs) operate on the principle of changes in electrical resistance of pure metals and are characterized by a linear positive change in resistance with temperature. Typical elements used for RTDs include nickel (Ni) and copper (Cu), but platinum (Pt) is by far the most common because of its wide temperature range, accuracy, and stability.

RTDs are constructed by one of two different manufacturing configurations. Wire-wound RTDs are constructed by winding a thin wire into a coil. A more common configuration is the thin-film element, which consists of a very thin layer of metal laid out on a plastic or ceramic substrate. Thin-film elements are cheaper and more widely available because they can achieve higher nominal resistances with less platinum. To protect the RTD, a metal sheath encloses the RTD element and the lead wires connected to it.

RTDs are popular because of their excellent stability, and exhibit the most linear signal with respect to temperature of any electronic temperature sensor. They are generally more expensive than alternatives, however, because of the careful construction and use of platinum. RTDs are also characterized by a slow response time and low sensitivity; and because they require current excitation, they can be prone to self-heating.

RTDs are commonly categorized by their nominal resistance at 0 °C. Typical nominal resistance values for platinum thin-film RTDs include 100 Ω and 1000 Ω. The relationship between resistance and temperature is very nearly linear and follows the equation

For <0 °C \( R_T = R_0 \left[ 1 + aT + bT^2 + cT^3 (T - 100) \right] \)

For >0 °C \( R_T = R_0 \left[ 1 + aT + bT^2 \right] \)
Where $R_T$ = resistance at temperature $T$

$R_0$ = nominal resistance

$a$, $b$, and $c$ are constants used to scale the RTD

The resistance/temperature curve for a 100 W platinum RTD, commonly referred to as Pt100, is shown below:

![Resistance-Temperature Curve for a 100 Ω Platinum RTD, a = 0.00385](image)

The most common RTD is the platinum thin-film with an $a$ of 0.385%/°C and is specified per DIN EN 60751. The $a$ value depends on the grade of platinum used, and also commonly include 0.3911%/°C and 0.3926%/°C. The $a$ value defines the sensitivity of the metallic element, but is normally used to distinguish between resistance/temperature curves of various RTDs.

### Callendar-Van Dusen Coefficients Corresponding to Common RTDs

<table>
<thead>
<tr>
<th>Standard</th>
<th>Temperature Coefficient (a)</th>
<th>$A$</th>
<th>$B$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIN 43760</td>
<td>0.003850</td>
<td>3.9080 x 10^{-3}</td>
<td>-5.8019 x 10^{-7}</td>
<td>-4.2735 x 10^{-12}</td>
</tr>
<tr>
<td>American</td>
<td>0.003911</td>
<td>3.9692 x 10^{-3}</td>
<td>-5.8495 x 10^{-7}</td>
<td>-4.2325 x 10^{-12}</td>
</tr>
<tr>
<td>ITS-90</td>
<td>0.003926</td>
<td>3.9848 x 10^{-3}</td>
<td>-5.870 x 10^{-7}</td>
<td>-4.0000 x 10^{-12}</td>
</tr>
</tbody>
</table>

* For temperatures below 0 °C only; $C = 0.0$ for temperatures above 0 °C.
1.3.2. Thermistors

Thermistors (thermally sensitive resistors) are similar to RTDs in that they are electrical resistors whose resistance changes with temperature. Thermistors are manufactured from metal oxide semiconductor material which is encapsulated in a glass or epoxy bead.

Thermistors have a very high sensitivity, making them extremely responsive to changes in temperature. For example, a 2252 W thermistor has a sensitivity of -100 W/°C at room temperature. In comparison, a 100 W RTD has a sensitivity of 0.4 W/°C. Thermistors also have a low thermal mass that results in fast response times, but are limited by a small temperature range.

Thermistors have either a negative temperature coefficient (NTC) or a positive temperature coefficient (PTC). The first has a resistance which decreases with increasing temperature and the latter exhibits increased resistance with increasing temperature. Figure 2 shows a typical thermistor temperature curve compared to a typical 100 W RTD temperature curve.

1.3.3. LM35

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies of $g/4\beta C$ at room temperature and $g^*/4\beta C$ over a full $b55$ to $a150\beta C$ temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single
power supplies, or with plus and minus supplies. As it draws only 60 mA from its supply, it has very low self-heating, less than 0.1°C in still air. The LM35 is rated to operate over a $-55^\circ C$ to $150^\circ C$ temperature range, while the LM35C is rated for a $-40^\circ C$ to $110^\circ C$ range ($b10^\circ C$ with improved accuracy). The LM35 series is available packaged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface mount small outline package.

Most commonly-used electrical temperature sensors are difficult to apply. For example, thermocouples have low output levels and require cold junction compensation. Thermistors are nonlinear. In addition, the outputs of these sensors are not linearly proportional to any temperature scale. Early monolithic sensors, such as the LM3911, LM134 and LM135, overcame many of these difficulties, but their outputs are related to the Kelvin temperature scale rather than the more popular Celsius and Fahrenheit scales. Fortunately, in 1983 two I.C.’s, the LM34 Precision Fahrenheit Temperature Sensor and the LM35 Precision Celsius Temperature Sensor, were introduced. The LM35 has an output of 10 mV/°F with a typical nonlinearity of only ±0.35°F over a $-50$ to $+300°F$ temperature range, and is accurate to within ±0.4°F typically at room temperature ($77°F$). The LM34’s low output impedance and linear output characteristic make interfacing with readout or control circuitry easy. An inherent strength of the LM34 over other currently available temperature sensors is that it is not as susceptible to large errors in its output from low level leakage currents. For instance, many monolithic temperature sensors have an output of only 1 μA/°K. This leads to a 1°K error for only 1 μ-Ampere of leakage current. On the other hand, the LM34 may be operated as a current mode device providing 20 μA/°F of output current. The same 1 μA of leakage current will cause an error in the LM34’s output of only 0.05°F (or 0.03°K after scaling). Low cost and high accuracy are maintained by performing trimming and calibration procedures at the wafer level. The device may be operated with either single or dual supplies. With less than 70 μA of current drain, the LM34 has
very little self-heating (less than 0.2°F in still air), and comes in a TO-46 metal can package, a SO-8 small outline package and a TO-92 plastic package.

LM35 Features

- Calibrated directly in ° Celsius (Centigrade)
- Linear + 10.0 mV/°C scale factor
- 0.5°C accuracy guaranteeable (at +25°C)
- Rated for full -55° to +150°C range
- Suitable for remote applications
- Low cost due to wafer-level trimming
- Less than 60 μA current drain
- Low self-heating, 0.08°C in still air
- Low impedance output, 0.