UNIT- 2

Author:- Dogan Ibrahim
"Advanced PIC Micro-C Projects in C".

USB :- Universal Serial Bus :-

Is one of the most common interface used in electronic consumer products today including PCs, Cameras, GPS Devices, MP3 players, modems, printers, and scanners.

→ The USB is a high-speed serial interface that can also provide power to device connected to it.

→ A USB bus supports upto 127 devices (limited by the 7-bit address field - note address '0' is not used as it has special purpose) connected to four wire serial cable upto three (or) even five meters in length.

→ The maximum distance of a device from its host is about 30 meters accomplished by using five hubs. For longer distance bus communications other methods such as use of Ethernet are recommended.

USB Specifications :-

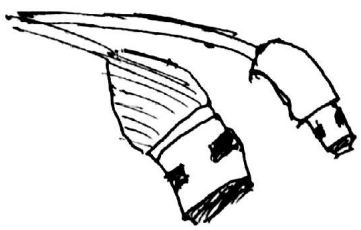
Comes in 2-versions
- USB 1.1 supports 11 MBPS
- USB 2.0 " 480 MBPS

They also define in 3-data speed.
- → low speed - 1.5 Mb/sec
- → Full speed - 12 Mb/sec
- → High speed - 480 Mb/sec

→ The Maximum power available to an external device is limited to about 100 mA at 5.0V

→ USB is a four wire implemented using a 4-core shielded cable.

→ Two types of connectors are specified
- (a) Type A
- (b) Type B.

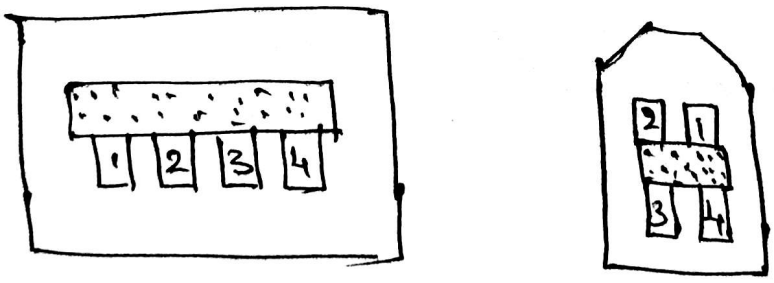


USB Connector pin assignments:-

Pin no:	Name	Color
1	+5V	RED
2	Data -	White
3	Data +	Green
4	Ground	Black

Mini USB pin assignments:-

Pin no:-	Name	Color
1	+5V	RED
2	Data -	White
3	Data +	Green
4	Not used	-
5	Ground	Black.



Pin-out USB Connectors.

USB signals are bi-phase, and signals are sent from the host computer using the NRZI (non-return to zero inverted) data encoding technique. In this technique, the signal level is inverted for each change to a logic 0. The signal level for a logic 1 is not changed. A 0 bit is "stuffed" after every six consecutive ones in the data stream to make the data dynamic (this is called *bit stuffing* because the extra bit lengthens the data stream). Figure 8.3 shows how the NRZI is implemented.

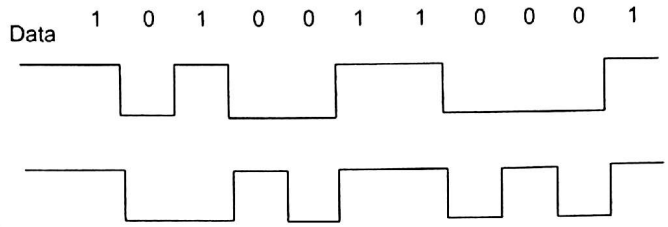


Figure 8.3: NRZI data

A packet of data transmitted by the host is sent to every device connected to the bus, traveling downward through the chain of hubs. All the devices receive the signal, but only one of them, the addressed one, accepts the data. Conversely, only one device at any time can transmit to the host, and the data travels upward through the chain of hubs until it reaches the host.

USB devices attached to the bus may be full-custom devices, requiring a full-custom device driver, or they may belong to a device class. Device classes enable the same device driver to be used for several devices having similar functionalities. For example, a printer device has the device class 0×07, and most printers use drivers of this type.

The most common device classes are given in Table 8.3. The USB human interface device (HID) class is of particular interest, as it is used in the projects in this chapter.

Some common USB terms are:

Endpoint: An endpoint is either a source or a sink of data. A single USB device can have a number of endpoints, the limit being sixteen IN and sixteen OUT endpoints.

Transaction: A transaction is a transfer of data on the bus.

Pipe: A pipe is a logical data connection between the host and an endpoint.

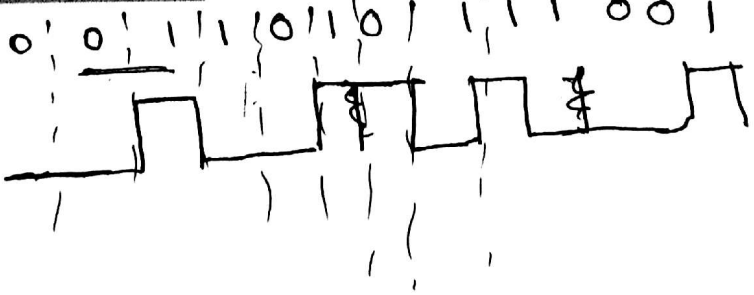


Table 8.3: USB device classes

Device class	Description	Example device
0×00	Reserved	–
0×01	USB audio device	Sound card
0×02	USB communications device	Modem, fax
0×03	USB human interface device	Keyboard, mouse
0×07	USB printer device	Printer
0×08	USB mass storage device	Memory card, flash drive
0×09	USB hub device	Hubs
0×0B	USB smart card reader device	Card reader
0×0E	USB video device	Webcam, scanner
0×E0	USB wireless device	Bluetooth

8.1 Speed Identification on the Bus

At the device end of the bus, a 1.5K pull-up resistor is connected from the D+ or D− line to 3.3V. On a full-speed bus, the resistor is connected from the D+ line to 3.3V, and on a low-speed bus the resistor is from D− line to 3.3V. When no device is plugged in, the host will see both data lines as low. Connecting a device to the bus will pull either the D+ or the D− line to logic high, and the host will know that a device is plugged into the bus. The speed of the device is determined by observing which line is pulled high.

8.2 USB States

Some of the USB bus states are:

Idle: The bus is in idle state when the pulled-up line is high and the other line is low. This is the state of the lines before and after a packet transmission.

Detached: When no device is connected to the bus, the host sees both lines as low.

Attached: When a device is connected to the bus, the host sees either D+ or D− go to logic high, which means a device has been plugged in.

J state: The same as idle state.

K state: The opposite of J state.

SE0: The single ended zero state, where both lines on the bus are pulled low.

SE1: The single ended one state, where both lines on the bus are high. SE1 is an illegal condition on the bus; it must never be in this state.

Reset: When the host wants to communicate with a device on the bus, it first sends a "reset" condition by pulling low both data lines (SE0 state) for at least 10ms.

EOP: The end of packet state, which is basically an SE0 state for 2 bit times, followed by a J state for 1 bit time.

Keep alive: The state achieved by EOP. Keep alive is sent at least once every millisecond to keep the device from suspending.

Suspend: Used to save power, suspend is implemented by not sending anything to a device for 3ms. A suspended device draws less than 0.5mA from the bus and must recognize reset and resume signals.

Resume: A suspended device is woken up by reversing the polarity of the signal on the data lines for at least 20ms, followed by a low-speed EOP signal.

8.3 USB Bus Communication

USB is a host-centric connectivity system where the host dictates the use of the USB bus. Each device on the bus is assigned a unique USB address, and no slave device can assert a signal on the bus until the host asks for it. When a new USB device is plugged into a bus, the USB host uses address 0 to ask basic information from the device. Then the host assigns it a unique USB address. After the host asks for and receives further information about the device, such as the name of the manufacturer, device capabilities, and product ID, two-way transactions on the bus can begin.

8.3.1 Packets

Data is transmitted on a USB bus in packets. A packet starts with a sync pattern to allow the receiver clock to synchronize with the data. The data bytes of the packet follow, ending with an end of packet signal.

A packet identifier (PID) byte immediately follows the sync field of every USB packet. A PID itself is 4 bits long, and the 4 bits are repeated in a complemented form. There are seventeen different PID values, as shown in Table 8.4. These include one reserved value and one that is used twice, with two different meanings.

There are four packet formats, based on which PID is at the start of the packet: token packets, data packets, handshake packets, and special packets.

Figure 8.4 shows the format of a token packet, which is used for OUT, IN, SOF (start of frame), and SETUP. The packet contains a 7-bit address, a 4-bit ENDP (endpoint number). a 5-bit CRC checksum, and an EOP (end of packet).

Table 8.4: PID values

PID type	PID name	Bits	Description
Token	OUT	1110 0001	Host to device transaction
	IN	0110 1001	Device to host transaction
	SOF	1010 0101	Start of frame
	SETUP	0010 1101	Setup command
Data	DATA0	1100 0011	Data packet PID even
	DATA1	0100 1011	Data packet PID odd
	DATA2	1000 0111	Data packet PID high speed
	MDATA	0000 1111	Data packet PID high speed
Handshake	ACK	1101 0010	Receiver accepts packet
	NAK	0101 1010	Receiver does not accept packet
	STALL	0001 1110	Stalled
	NYET	1001 0110	No response from receiver
Special	PRE	0011 1100	Host preample
	ERR	0011 1100	Split transaction error
	SPLIT	0111 1000	High-speed split transaction
	PING	1011 0100	High-speed flow control
	Reserved	1111 0000	Reserved

Sync	PID	ADDR	ENDP	CRC	EOP
	8 bits	7 bits	4 bits	5 bits	

Figure 8.4: Token packet

A data packet is used for DATA0, DATA1, DATA2, and MDATA data transactions. The packet format is shown in Figure 8.5 and consists of the PID, 0–1024 bytes of data, a 2-byte CRC checksum, and an EOP.

Sync	PID	Data	CRC	EOP
	1 byte	0–1024 bytes	2 bytes	

Figure 8.5: Data packet

Sync	PID	EOP
	1 byte	

Figure 8.6: Handshake packet

Figure 8.6 shows the format of a handshake packet, which is used for ACK, NAK, STALL, and NYET. ACK is used when a receiver acknowledges that it has received an error-free data packet. NAK is used when the receiving device cannot accept the packet. STALL indicates when the endpoint is halted, and NYET is used when there is no response from the receiver.

8.3.2 Data Flow Types

Data can be transferred on a USB bus in four ways: bulk transfer, interrupt transfer, isochronous transfer, and control transfer.

1. *Bulk transfers* are designed to transfer large amounts of data with error-free delivery and no guarantee of bandwidth. If an OUT endpoint is defined as using bulk transfers, then the host will transfer data to it using OUT transactions. Similarly, if an IN endpoint is defined as using bulk transfers, then the host will transfer data from it using IN transactions. In general, bulk transfers are used where a slow rate of transfer is not a problem. The maximum packet size in a bulk transfer is 8 to 64 packets at full speed, and 512 packets at high speed (bulk transfers are not allowed at low speeds).

2. *Interrupt transfers* are used to transfer small amounts of data with a high bandwidth where the data must be transferred as quickly as possible with no delay. Note that interrupt transfers have nothing to do with interrupts in computer systems. Interrupt packets can range in size from 1 to 8 bytes at low speed, from 1 to 64 bytes at full speed, and up to 1024 bytes at high speed.

3. *Isochronous transfers* have a guaranteed bandwidth, but error-free delivery is not guaranteed. This type of transfer is generally used in applications, such as audio data

transfer, where speed is important but the loss or corruption of some data is not. An isochronous packet may contain 1023 bytes at full speed or up to 1024 bytes at high speed (isochronous transfers are not allowed at low speeds).

A *control transfer* is a bidirectional data transfer, using both IN and OUT endpoints. Control transfers are generally used for initial configuration of a device by the host. The maximum packet size is 8 bytes at low speed, 8 to 64 bytes at full speed, and 64 bytes at high speed. A control transfer is carried out in three stages: SETUP, DATA, and STATUS.

8.3.3 Enumeration

When a device is plugged into a USB bus, it becomes known to the host through a process called enumeration. The steps of enumeration are:

- When a device is plugged in, the host becomes aware of it because one of the data lines (D+ or D−) becomes logic high.
- The host sends a USB reset signal to the device to place the device in a known state. The reset device responds to address 0.
- The host sends a request on address 0 to the device to find out its maximum packet size using a *Get Descriptor* command.
- The device responds by sending a small portion of the device descriptor.
- The host sends a USB reset again.
- The host assigns a unique address to the device and sends a *Set Address* request to the device. After the request is completed, the device assumes the new address. At this point the host is free to reset any other newly plugged-in devices on the bus.
- The host sends a *Get Device Descriptor* request to retrieve the complete device descriptor, gathering information such as manufacturer, type of device, and maximum control packet size.
- The host sends a *Get Configuration Descriptors* request to receive the device's configuration data, such as power requirements and the types and number of interfaces supported.
- The host may request any additional descriptors from the device.

The initial communication between the host and the device is carried out using the control transfer type of data flow.

Initially, the device is addressed, but it is in an unconfigured state. After the host gathers enough information about the device, it loads a suitable device driver which configures the device by sending it a *Set Configuration* request. At this point the device has been configured, and it is ready to respond to device-specific requests (i.e., it can receive data from and send data to the host).

8.4 Descriptors

All USB devices have a hierarchy of descriptors that describe various features of the device: the manufacturer ID, the version of the device, the version of USB it supports, what the device is, its power requirements, the number and type of endpoints, and so forth.

The most common USB descriptors are:

- Device descriptors
- Configuration descriptors
- Interface descriptors
- HID descriptors
- Endpoint descriptors

The descriptors are in a hierarchical structure as shown in Figure 8.7. At the top of the hierarchy we have the device descriptor, then the configuration descriptors, followed by the interface descriptors, and finally the endpoint descriptors. The HID descriptor always follows the interface descriptor when the interface belongs to the HID class.

All descriptors have a common format. The first byte (*bLength*) specifies the length of the descriptor, while the second byte (*bDescriptorType*) indicates the descriptor type.

8.4.1 Device Descriptors

The device descriptor is the top-level set of information read from a device and the first item the host attempts to retrieve.

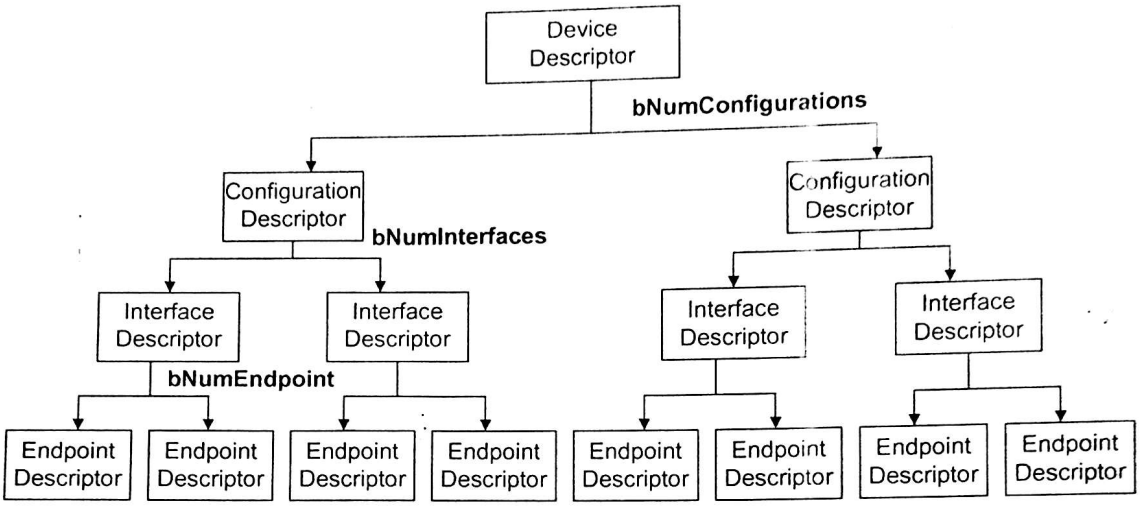


Figure 8.7: USB descriptor hierarchy

A USB device has only one device descriptor, since the device descriptor represents the entire device. It provides general information such as manufacturer, serial number, product number, the class of the device, and the number of configurations. Table 8.5 shows the format for a device descriptor with the meaning of each field.

bLength is the length of the device descriptor.

bDescriptorType is the descriptor type.

bcdUSB reports the highest version of USB the device supports in BCD format. The number is represented as 0×JJMN, where JJ is the major version number, M is the minor version number, and N is the subminor version number. For example, USB 1.1 is reported as 0×0110.

bDeviceClass, *bDeviceSubClass*, and *bDeviceProtocol* are assigned by the USB organization and are used by the system to find a class driver for the device.

bMaxPacketSize0 is the maximum input and output packet size for endpoint 0.

idVendor is assigned by the USB organization and is the vendor's ID.

idProduct is assigned by the manufacturer and is the product ID.

bcdDevice is the device release number and has the same format as the *bcdUSB*.

Table 8.5: Device descriptor

Offset	Field	Size	Description
0	bLength	1	Descriptor size in bytes
1	bDescriptorType	1	Device descriptor (0×01)
2	bcdUSB	2	Highest version of USB supported
4	bDeviceClass	1	Class code
5	bDeviceSubClass	1	Subclass code
6	bDeviceProtocol	1	Protocol code
7	bMaxPacketSize0	1	Maximum packet size
8	idVendor	2	Vendor ID
10	idProduct	2	Product ID
12	bcdDevice	2	Device release number
14	iManufacturer	1	Manufacturer string descriptor
15	iProduct	1	Index of product string descriptor
16	iSerialNumber	1	Index of serial number descriptor
17	bNumConfigurations	1	Number of possible configurations

iManufacturer, *iProduct*, and *iSerialNumber* are details about the manufacturer and the product. These fields have no requirement and can be set to zero.

bNumConfigurations is the number of configurations the device supports.

Table 8.6 shows an example device descriptor for a mouse device. The length of the descriptor is 18 bytes (*bLength* = 18), and the descriptor type is 0×01 (*bDescriptorType* = 0×01). The device supports USB 1.1 (*bcdUSB* = 0×0110). *bDeviceClass*, *bDeviceSubClass*, and *bDeviceProtocol* are set to zero to show that the class information is in the interface descriptor. *bMaxPacketSize0* is set to 8 to show that the maximum input and output packet size for endpoint 0 is 8 bytes. The next three bytes identify the device by the vendor ID, product ID, and device version number. The next three items define indexes to strings about the manufacturer, product, and the serial number. Finally, we notice that the mouse device has just one configuration (*bNumConfigurations* = 1).

Table 8.6: Example device descriptor

Offset	Field	Value	Description
0	bLength	18	Size is 18 bytes
1	bDescriptorType	0×01	Descriptor type
2	bcdUSB	0×0110	Highest USB supported = USB 1.1
4	bDeviceClass	0×00	Class information in interface descriptor
5	bDeviceSubClass	0×00	Class information in interface descriptor
6	bDeviceProtocol	0×00	Class information in interface descriptor
7	bMaxPacketSize0	8	Maximum packet size
8	idVendor	0×02A	XYZ Co Ltd.
10	idProduct	0×1001	Mouse
12	bcdDevice	0×0011	Device release number
14	iManufacturer	0×20	Index to manufacturer string
15	iProduct	0×21	Index of product string
16	iSerialNumber	0×22	Index of serial number string
17	bNumConfigurations	1	Number of possible configurations

8.4.2 Configuration Descriptors

The configuration descriptor provides information about the power requirements of the device and how many different interfaces it supports. There may be more than one configuration for a device.

Table 8.7 shows the format of the configuration descriptor with the meaning of each field.

bLength is the length of the device descriptor.

bDescriptorType is the descriptor type.

wTotalLength is the total combined size of this set of descriptors (i.e., total of configuration descriptor + interface descriptor + HID descriptor + endpoint descriptor). When the configuration descriptor is read by the host, it returns the entire configuration information, which includes all interface and endpoint descriptors.

Table 8.7: Configuration descriptor

Offset	Field	Size	Description
0	bLength	1	Descriptor size in bytes
1	bDescriptorType	1	Device descriptor (0×02)
2	wTotalLength	2	Total bytes returned
4	bNumInterfaces	1	Number of interfaces
5	bConfigurationValue	1	Value used to select configuration
6	iConfiguration	1	Index describing configuration string
7	bmAttributes	1	Power supply attributes
8	bMaxPower	2	Max power consumption in 2mA

bNumInterfaces is the number of interfaces present for this configuration.

bConfigurationValue is used by the host (in command *SetConfiguration*) to select the configuration.

iConfiguration is an index to a string descriptor describing the configuration in readable format.

bmAttributes describes the power requirements of the device. If the device is USB bus-powered, then bit D7 is set. If it is self-powered, it sets bit D6. Bit D5 specifies the remote wakeup of the device. Bits D7 and D0–D4 are reserved.

bMaxPower defines the maximum power the device will draw from the bus in 2mA units.

Table 8.8 shows an example configuration descriptor for a mouse device. The length of the descriptor is 9 bytes (*bLength* = 9), and the descriptor type is 0×02 (*bDescriptorType* = 0×02). The total combined size of the descriptors is 34 (*wTotalLength* = 34). The number of interfaces for the mouse device is 1 (*bNumInterfaces* = 1). *Host SetConfiguration* command must use the value 1 as an argument in *SetConfiguration()* to select this configuration. There is no string to describe this configuration. *bmAttributes* is set to 0×40 to indicate that the device is self-powered. *bMaxPower* is set to 10 to specify that the maximum current drawn by the device is 20mA.

Table 8.8: Example configuration descriptor

Offset	Field	Value	Description
0	bLength	9	Descriptor size is 9 bytes
1	bDescriptorType	0×02	Device descriptor is 0×02
2	wTotalLength	34	Total bytes returned is 34
4	bNumInterfaces	1	Number of interfaces is 1
5	bConfigurationValue	1	Value used to select configuration
6	iConfiguration	0×2A	Index describing configuration string
7	bmAttributes	0×40	Power supply attributes
8	bMaxPower	10	Max power consumption is 20mA

8.4.3 Interface Descriptors

The interface descriptors specify the class of the interface and the number of endpoints it uses. There may be more than one interface.

Table 8.9 shows the format of the interface descriptor with the meaning of each field.

Table 8.9: Interface descriptor

Offset	Field	Size	Description
0	bLength	1	Descriptor size in bytes
1	bDescriptorType	1	Device descriptor (0×04)
2	bInterfaceNumber	1	Number of interface
3	bAlternateSetting	1	Value to select alternate setting
4	bNumEndpoints	1	Number of endpoints
5	bInterfaceClass	1	Class code
6	bInterfaceSubClass	1	Subclass code
7	bInterfaceProtocol	1	Protocol code
8	iInterface	1	Index of string descriptor to interface

bLength is the length of the device descriptor.

bDescriptorType is the descriptor type.

bInterfaceNumber indicates the index of the interface descriptor.

bAlternateSetting can be used to specify alternate interfaces that can be selected by the host using command *Set Interface*.

bNumEndpoints indicates the number of endpoints used by the interface.

bInterfaceClass specifies the device class code (assigned by the USB organization).

bInterfaceSubClass specifies the device subclass code (assigned by the USB organization).

bInterfaceProtocol specifies the device protocol code (assigned by the USB organization).

iInterface is an index to a string descriptor of the interface.

Table 8.10 shows an example interface descriptor for a mouse device. The descriptor length is 9 bytes (*bLength* = 9) and the descriptor type is 0×04 (*bDescriptorType* = 0×04). The interface number used to reference this interface is 1 (*bInterfaceNumber* = 1).

Table 8.10: Example interface descriptor

Offset	Field	Value	Description
0	bLength	9	Descriptor size is 9 bytes
1	bDescriptorType	0×04	Device descriptor is 0×04
2	bInterfaceNumber	0	Number of interface
3	bAlternateSetting	0	Value to select alternate setting
4	bNumEndpoints	1	Number of endpoints is 1
5	bInterfaceClass	0×03	Class code is 0×03
6	bInterfaceSubClass	0×02	Subclass code is 0×02
7	bInterfaceProtocol	0×02	Protocol code is 0×02
8	iInterface	0	Index of string descriptor to interface

bAlternateSetting is set to 0 (i.e., no alternate interfaces). The number of endpoints used by this interface is 1 (excluding endpoint 0), and this is the endpoint used for the mouse to send its data. The device class code is 0×03 (*bInterfaceClass* = 0×03). This is an HID (human interface device) type class. The interface subclass is set to 0×02. The device protocol is 0×02 (mouse). There is no string to describe this interface (*iInterface* = 0).

8.4.4 HID Descriptors

An HID descriptor always follows an interface descriptor when the interface belongs to the HID class. Table 8.11 shows the format of the HID descriptor.

bLength is the length of the device descriptor.

bDescriptorType is the descriptor type.

bcdHID is the HID class specification.

bCountryCode specifies any special local changes.

bNumDescriptors specifes if there are any additional descriptors associated with this class.

bDescriptorType is the type of the additional descriptor specified in *bNumDescriptors*.

wDescriptorLength is the length of the additional descriptor in bytes.

Table 8.11: HID descriptor

Offset	Field	Size	Description
0	bLength	1	Descriptor size in bytes
1	bDescriptorType	1	HID (0×21)
2	bcdHID	2	HID class
4	bCountryCode	1	Special country dependent code
5	bNumDescriptors	1	Number of additional descriptors
6	bDescriptorType	1	Type of additional descriptor
7	wDescriptorLength	2	Length of additional descriptor

Table 8.12 shows an example HID descriptor for a mouse device. The length of the descriptor is 9 bytes (*bLength* = 9), and the descriptor type is 0×21 (*bDescriptorType* = 0×21). The HID class is set to 1.1 (*bcdHID* = 0×0110). The country code is set to zero (*bCountryCode* = 0), specifying that there is no special localization with this device. The number of descriptors is set to 1 (*bNumDescriptors* = 1) which specifies that there is one additional descriptor associated with this class. The type of the additional descriptor is REPORT (*bDescriptorType* = REPORT), and its length is 52 bytes (*wDescriptorLength* = 52).

Table 8.12: Example HID descriptor

Offset	Field	Value	Description
0	bLength	9	Descriptor size is 9 bytes
1	bDescriptorType	0×21	HID (0×21)
2	bcdHID	0×0110	Class version 1.1
4	bCountryCode	0	No special country dependent code
5	bNumDescriptors	1	Number of additional descriptors
6	bDescriptorType	REPORT	Type of additional descriptor
7	wDescriptorLength	5	Length of additional descriptor

8.4.5 Endpoint Descriptors

Table 8.13 shows the format of the endpoint descriptor.

bLength is the length of the device descriptor.

bDescriptorType is the descriptor type.

bEndpointAddress is the address of the endpoint.

bmAttributes specifies what type of endpoint it is.

wMaxPacketSize is the maximum packet size.

bInterval specifies how often the endpoint should be polled (in ms).

Table 8.14 shows an example endpoint descriptor for a mouse device. The length of the descriptor is 7 bytes (*bLength* = 7), and the descriptor type is 0×05 (*bDescriptorType*

Table 8.13: Endpoint descriptor

Offset	Field	Size	Description
0	bLength	1	Descriptor size in bytes
1	bDescriptorType	1	Endpoint (0×05)
2	bcdEndpointAddress	1	Endpoint address
4	bmAttributes	1	Type of endpoint
5	wMaxPacketSize	2	Max packet size
6	bInterval	1	Polling interval

Table 8.14: Example endpoint descriptor

Offset	Field	Size	Description
0	bLength	7	Descriptor size in bytes
1	bDescriptorType	0×05	Endpoint (0×05)
2	bcdEndpointAddress	0×50	Endpoint address
4	bmAttributes	0×03	Interrupt type endpoint
5	wMaxPacketSize	0×0002	Max packet size is 2
6	bInterval	0×14	Polling interval is 20ms

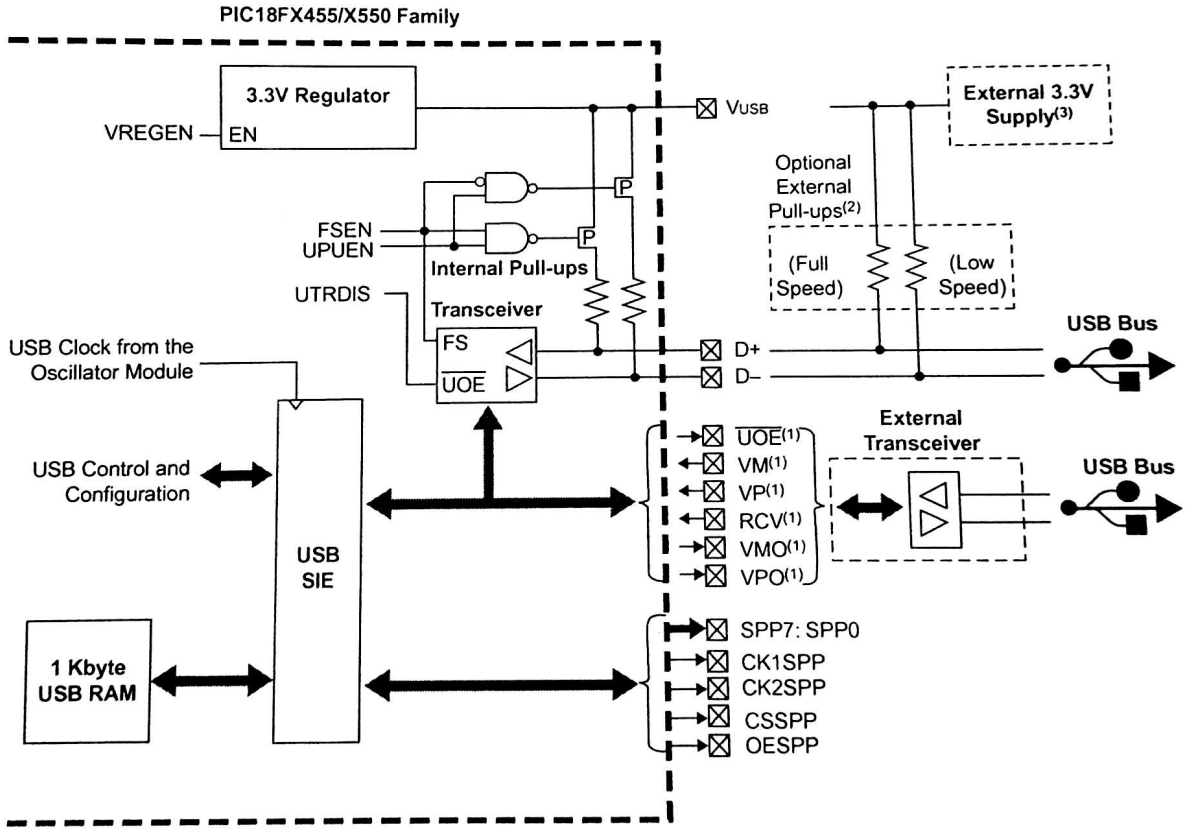
$= 0\times05$). The endpoint address is 0×50 (*bEndpointAddress* $= 0\times50$). The endpoint is to be used as an interrupt endpoint (*bmAttributes* $= 0\times03$). The maximum packet size is set to 2 (*wMaxPacketSize* $= 0\times02$) to indicate that packets longer than 2 bytes will not be sent from the endpoint. The endpoint should be polled at least once every 20ms (*bInterval* $= 0\times14$).

8.5 PIC18 Microcontroller USB Bus Interface

Some of the PIC18 microcontrollers support USB interface directly. For example, the PIC18F4550 microcontroller contains a full-speed and low-speed compatible USB interface that allows communication between a host PC and the microcontroller. In the USB projects in this chapter we will use the PIC18F4550 microcontroller.

Figure 8.8 is an overview of the USB section of the PIC18F4550 microcontroller. PORTC pins RC4 (pin 23) and RC5 (pin 24) are used for USB interface. RC4 is the USB data D− pin, and RC5 is the USB data D+ pin. Internal pull-up resistors are provided which can be disabled (setting *UPUEN* = 0) if desired and external pull-up resistors can be used instead. For full-speed operation an internal or external resistor should be connected to data pin D+, and for low-speed operation an internal or external resistor should be connected to data pin D−.

Operation of the USB module is configured using three control registers, and a total of twenty-two registers are used to manage the actual USB transactions. Configuration



Note 1: This signal is only available if the internal transceiver is disabled (UTRDIS = 1).
2: The internal pull-up resistors should be disabled (UPUEN = 0) if external pull-up resistors are used.
3: Do not enable the internal regulator when using an external 3.3V supply.

Figure 8.8: PIC18F4550 microcontroller USB overview

of these registers is a highly complex task and is not covered in this book. Interested readers should refer to the PIC18F4550 data sheet and to books on USB internals. In this chapter we are using the mikroC language USB library functions to implement USB transactions. The details of these functions are given in the next section.

8.6 mikroC Language USB Bus Library Functions

The mikroC language supports a number of functions for USB HID-type communications. Each project based on the USB library should include a descriptor source file which contains vendor ID and name, product ID and name, report length, and other relevant information. To create a descriptor source file we can use mikroC's integrated USB HID terminal tool (see *Tools → HID Terminal*). The default name for descriptor file is *USBdsc.c*, but it can be renamed if required. The *USBdsc.c* file must be included in USB-based projects either via the mikroC IDE tool, or as an *#include* option in the program source file.

The mikroC language supports the following USB bus library functions when a PIC microcontroller with built-in USB is used (e.g., PIC18F4550), and port pins RC4 and RC5 are connected to the D+ and D− pins of the USB connector respectively:

Hid_Enable: This function enables USB communication and requires two arguments: the read-buffer address and the write-buffer address. It must be called before any other functions of the USB library, and it returns no data.

Hid_Read: This function receives data from the USB bus and stores it in the receive-buffer. It has no arguments but returns the number of characters received.

Hid_Write: This function sends data from the write-buffer to the USB bus. The name of the buffer (the same buffer used in the initialization) and the length of the data to be sent must be specified as arguments to the function. The function does not return any data.

Hid_Disable: This function disables the USB data transfer. It has no arguments and returns no data.

The USB interface of a PIC18F4550 microcontroller is shown in Figure 8.9. As the figure shows, the interface is very simple. In addition to the power supply and ground pins, it requires just two pins to be connected to the USB connector. The microcontroller receives power from the USB port.

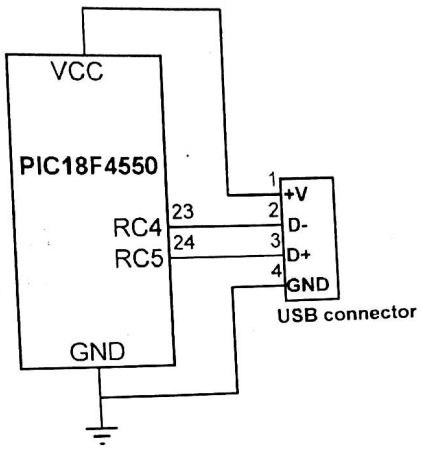


Figure 8.9: PIC18F4550 USB interface

PROJECT 8.1—USB-Based Microcontroller Output Port

This project describes the design of a USB-based microcontroller output port. A PIC18F4550 microcontroller is interfaced to a PC through a USB cable. A Visual Basic program runs on the PC and sends commands to the microcontroller through the USB bus, asking the microcontroller to set/reset the I/O bits of its PORTB.

The block diagram of the project is shown in Figure 8.10. The circuit diagram is given in Figure 8.11. The USB lines of the PIC18F4550 microcontroller are connected to a USB connector. The microcontroller is powered from the USB line (i.e., no external

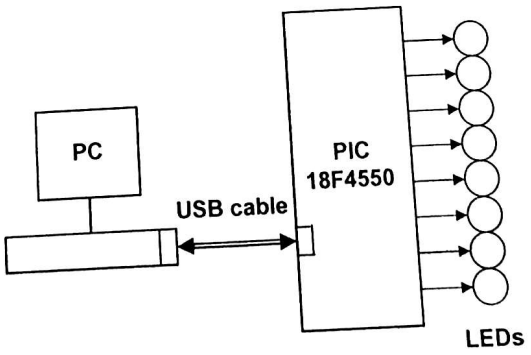


Figure 8.10: Block diagram of the project

2) USB Data flow types :- (4-types)

(i) → Bulk Transfer :- max packet size in a bulk transfer is 8 to 64 packets at full speed and 512 packets at high speed

(ii) → Interrupt transfer :- ranges in size from (1 to 8 bytes) at low speed and (1 to 64) bytes at full speed upto (1024 bytes) at high speed.

(iii) → Isochronous transfer :- (error free delivery) used application as audio data transfer. (Isochronous packets contains 1023 bytes @ full speed upto 1024 bytes at high speed.)

(iv) A Control transfer :- is a bidirectional data transfer. (max packet size is 8-bytes at low speed 8 to 64 bytes at full speed and 64-bytes at high speed.) → Control is 3-stages SETUP ; DATA ; STATUS.

3) Enumeration :- The process of knowing to host when a device is plugged in USB Bus called as Enumeration.

Step 1 :- Device plugged in the host becomes aware of device plugged by (D+/D-) becomes logic high

Step 2 :- Host sends usb reset signal. the reset device responds to address '0'.

Step 3 :- Host " a Request on address 0 to find maxm packet size using a Get Descriptor command.

Step 4 :- Devices responds by small portion of device descriptor.

Step 5 :- Host sends usb reset again.

Step 6 :- Host assign a unique address to device and sends set Address request to device.

Step 7 :- After request completed devices assume new address at this point host is free to reset any other newly plugged in device on bus.

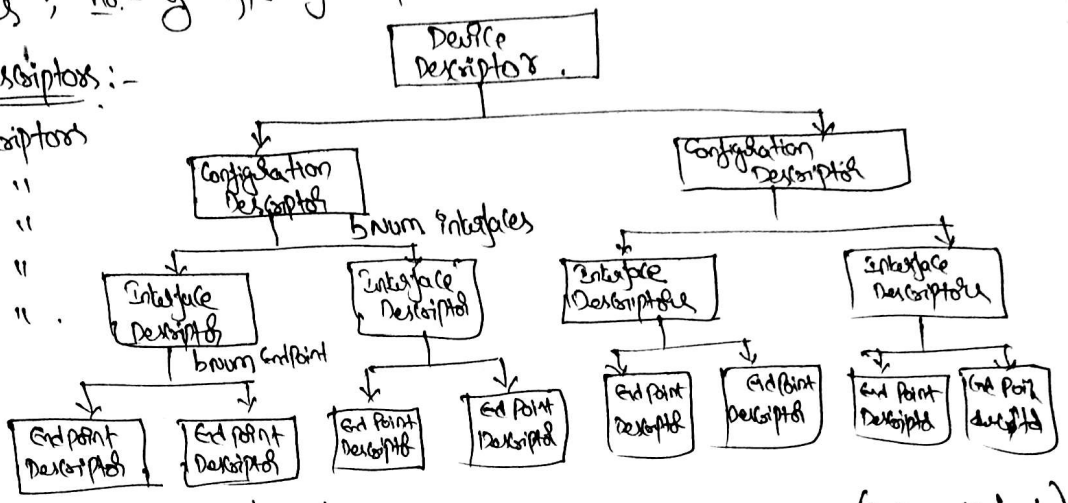
Step 8 :- Host sends get device Descriptor request to retrive the complete device descriptor. gather information such as Manufacturer, type of device, max Control packet size.

Step 9 :- Host sends a Get Configuration Descriptor request to receive the devices Configurations data such as power requirements, types and no:- of interfaces supported.

Step 10 :- Host may request any additional descriptors from the device.

4) **Descriptors**:- USB devices have a hierarchy of descriptors describe various features:-
→ Manufacturer ID, version of device; version of USB it supports; Power requirements; no:- of type of endpoints.

Common USB Descriptors:-
ⓐ Device descriptors
ⓑ Configuration "
ⓒ Interface "
ⓓ HID "
ⓔ Endpoint " .



All descriptors have common format
→ First byte (b length) specifies the length of descriptor, while second byte (b Descriptor length)

ⓐ **Device Descriptors**:- Top level Descriptor set of information read from a device and host attempts to retrieve.
→ A USB device has only one device descriptor, since it represents entire device. to provide general information like Manufacture, Serial no:-, Product number; class of device; no:- of configurations.
b length is the length of device descriptor
b Descriptor type is descriptor type.
→ bcd USB reports highest version of USB which support USB format. The device is represented as 0xJJMN, where JJ is major version number, M. is minor version number; N. is subminor version number;
Eg:- USB 1.1 is reported as 0x0110.
→ b Device class, b Device subclass, and b Device protocol are assigned by USB organization and are used by system to find a class driver for device.
→ b Max packet size 0 is maximum i/p & o/p packet size for Endpoint 0.
→ id vendor is assigned by the USB organization and is the vendor's ID.
→ id Product " " " manufacturer and is Product ID.
→ bcd Device is device release number and has same format bcd USB.