Beyond Logic

USB in a NutShell

Making sense of the USB standard

The Setup Packet

Every USB device must respond to setup packets on the default pipe. The setup pa
used for detection and configuration of the device and carry out common functions s
setting the USB device's address, requesting a device descriptor or checking the st
endpoint.

A USB compliant Host expects all requests to be processed within a maximum peri
seconds. It also specifies stricter timing for specific requests :

- Standard Device requests without a data stage must be completed in 50ms.
- Standard Device requests with a data stage must start to return data 500ms a
request.
  - Each data packet must be sent within 500ms of the successful transmiss
previous packet.
  - The status stage must complete within 50ms after the transmission of th
packet.
- The SetAddress command (which contains a data phase) must process the co
return status within 50ms. The device then has 2ms to change address before
request is sent.

These timeout periods are quite acceptable for even the slowest of devices, but car
restriction during debugging. 50mS doesn't provide for many debugging characters
9600bps on an asynchronous serial port or for a In Circuit Debugger/Emulator to sir
to break execution to examine the internal Registers. As a result, USB requires som
debugging methods to that of other microcontroller projects.

*Casually reading through the XP DDK, one may note the Host Controller Drive
USBUSER_OP_SEND_ONE_PACKET command which is commented to rea
is used to implement the 'single step' USB transaction development tool." Whi
tool has not been released yet, we can only hope to see one soon.*

Each request starts with a 8 byte long Setup Packet which has the following format,

Offset	Field	Size	Value	Description
0	bmRequestType	1	Bit-Map	**D7 Data Phase Transfer Direction** 0 = Host to Device 1 = Device to Host **D6..5 Type** 0 = Standard 1 = Class 2 = Vendor 3 = Reserved **D4..0 Recipient** 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4..31 = Reserved
1	bRequest	1	Value	Request
2	wValue	2	Value	Value
4	wIndex	2	Index or Offset	Index
6	wLength	2	Count	Number of bytes to transfer if there is a data phase

The **bmRequestType** field will determine the direction of the request, type of reque designated recipient. The **bRequest** field determines the request being made. The bmRequestType is normally parsed and execution is branched to a number of hand a Standard Device request handler, a Standard Interface request handler, a Standa request handler, a Class Device request handler etc. How you parse the setup pack up to your preference. Others may choose to parse the bRequest first and then dete type and recipient based on each request.

Standard requests are common to all USB devices and are detailed in the next com Class requests are common to classes of drivers. For example, all device conformir class will have a common set of class specific requests. These will differ to a device to the communications class and differ again to that of a device conforming to the m class.

And last of all is the vendor defined requests. These are requests which you as the designer can assign. These are normally different from device to device, but this is a implementation and imagination.

A common request can be directed to different recipients and based on the recipien different functions. A GetStatus Standard request for example, can be directed at th interface or endpoint. When directed to a device it returns flags indicating the status wakeup and if the device is self powered. However if the same request is directed a interface it always returns zero, or should it be directed at an endpoint will return the the endpoint.

The **wValue** and **wIndex** fields allow parameters to be passed with the request. **wL** used the specify the number of bytes to be transferred should there be a data phase

Standard Requests

Section 9.4 of the USB specification details the "Standard Device" requests required implemented for every USB device. The standard provides a single table grouping i request. Considering most firmware will parse the setup packet by recipient we will up the requests based by recipient for easier examination and implementation.

Standard Device Requests

There are currently eight Standard Device requests, all of which are detailed in the t

bmRequestType	bRequest	wValue	wIndex	wLength	
1000 0000b	GET_STATUS (0x00)	Zero	Zero	Two	
0000 0000b	CLEAR_FEATURE (0x01)	Feature Selector	Zero	Zero	
0000 0000b	SET_FEATURE (0x03)	Feature Selector	Zero	Zero	
0000 0000b	SET_ADDRESS (0x05)	Device Address	Zero	Zero	
1000 0000b	GET_DESCRIPTOR (0x06)	Descriptor Type & Index	Zero or Language ID	Descriptor Length	D
0000 0000b	SET_DESCRIPTOR (0x07)	Descriptor Type & Index	Zero or Language ID	Descriptor Length	D
1000 0000b	GET_CONFIGURATION (0x08)	Zero	Zero	1	Cc
0000 0000b	SET_CONFIGURATION (0x09)	Configuration Value	Zero	Zero	

- *The **Get Status** request directed at the device will return two bytes during the with the following format,*

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1
Reserved														Remo Wake

If D0 is set, then this indicates the device is self powered. If clear, the device i powered. If D1 is set, the device has remote wakeup enabled and can wake tl during suspend. The remote wakeup bit can be by the SetFeature and ClearF *requests with a feature selector of DEVICE_REMOTE_WAKEUP (0x01)*

- ***Clear Feature** and **Set Feature** requests can be used to set boolean features designated recipient is the device, the only two feature selectors available are DEVICE_REMOTE_WAKEUP and TEST_MODE. Test mode allows the devic various conditions. These are further documented in the USB Specification Re*

- ***Set Address** is used during enumeration to assign a unique address to the U The address is specified in wValue and can only be a maximum of 127. This r unique in that the device does not set its address until after the completion of stage. (See Control Transfers.) All other requests must complete before the s*

- ***Set Descriptor/Get Descriptor** is used to return the specified descriptor in w request for the configuration descriptor will return the device descriptor and al and endpoint descriptors in the one request.*
  - *Endpoint Descriptors cannot be accessed directly by a GetDescriptor/Se Request.*
  - *Interface Descriptors cannot be accessed directly by a GetDescriptor/Se Request.*
  - *String Descriptors include a Language ID in wIndex to allow for multiple support.*

- ***Get Configuration/Set Configuration** is used to request or set the current de configuration. In the case of a Get Configuration request, a byte will be return data stage indicating the devices status. A zero value means the device is not and a non-zero value indicates the device is configured. Set Configuration is u enable a device. It should contain the value of bConfigurationValue of the des configuration descriptor in the lower byte of wValue to select which configurat*

Standard Interface Requests

The specification current defines five Standard Interface requests which are detailed below. Interestingly enough, only two requests do anything intelligible.

bmRequestType	bRequest	wValue	wIndex	wLength
1000 0001b	GET_STATUS (0x00)	Zero	Interface	Two
0000 0001b	CLEAR_FEATURE (0x01)	Feature Selector	Interface	Zero
0000 0001b	SET_FEATURE (0x03)	Feature Selector	Interface	Zero
1000 0001b	GET_INTERFACE (0x0A)	Zero	Interface	One
0000 0001b	SET_INTERFACE (0x11)	Alternative Setting	Interface	Zero

- ***wIndex** is normally used to specify the referring interface for requests directec interface. Its format is shown below.*

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2
Reserved								Interface Number					

- ***Get Status** is used to return the status of the interface. Such a request to the should return two bytes of 0x00, 0x00. (Both bytes are reserved for future use*
- ***Clear Feature** and **Set Feature** requests can be used to set boolean features designated recipient is the interface, the current USB Specification Revision 2 interface features.*
- ***Get Interface** and **Set Interface** set the Alternative Interface setting which is more detail under the Interface Descriptor.*

Standard Endpoint Requests

Standard Endpoint requests come in the four varieties listed below.

bmRequestType	bRequest	wValue	Windex	wLength	
1000 0010b	GET_STATUS (0x00)	Zero	Endpoint	Two	E
0000 0010b	CLEAR_FEATURE (0x01)	Feature Selector	Endpoint	Zero	
0000 0010b	SET_FEATURE (0x03)	Feature Selector	Endpoint	Zero	
1000 0010b	SYNCH_FRAME (0x12)	Zero	Endpoint	Two	Fra

- *The **wIndex** field is normally used to specify the referring endpoint and directi requests directed to an endpoint. Its format is shown below.*

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2
Reserved								Dir	Reserved			Endpo	

- ***Get Status** returns two bytes indicating the status (Halted/Stalled) of a endpo format of the two bytes returned is illustrated below.*

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2
Reserved													

- ***Clear Feature*** *and* ***Set Feature*** *are used to set Endpoint Features. The stanc defines one endpoint feature selector, ENDPOINT_HALT (0x00) which allows stall and clear an endpoint. Only endpoints other than the default endpoint is recommended to have this functionality.*

- *A* ***Synch Frame*** *request is used to report an endpoint synchronisation frame.*

Chapter 5 : USB Descriptors

- Device Descriptors
- Configuration Descriptors
- Interface Descriptors
- Endpoint Descriptors
- String Descriptors

Chapter 7 : Example Firmwa

- Enumeration
- Firmware Example - PDIUSBD1
- Source Code

Comments and Feedback?

Comments :
Email Address : (Optional) Send