FT8U232 / FT8U245 DEVICES
Latency and Data Throughput

Background

The Universal Serial Bus may be new to some users and developers. This application note tries to describe the major architecture differences that need to be considered by both software and hardware designers when changing from a traditional RS232 based solution, to one that uses the FT8U232 USB to serial device.

The need for handshaking

USB data transfer is prone to delays that do not normally appear in systems that have been used to transferring data using interrupts. The original COM ports of a PC were interrupt driven and therefore data could be transferred without using handshaking and still arrive into the PC without data loss. USB does not transfer data using interrupts. It uses a scheduled system and as a result, there can be periods when the USB request does not get scheduled and, if handshaking is not used, data loss will occur. An example of scheduling delays can be seen if an open application is dragged around using the mouse.

With a USB device, the method of transfer is done by packets. If data is to be sent from the PC, then a packet of data is built up by the device driver and sent to the USB scheduler. This scheduler puts the request onto the list of tasks for the USB host controller to perform. This will typically take at least 1 millisecond to execute because it will not pick up the new request until the next ‘ USB Frame’ (The frame period is 1 millisecond). There is therefore a sizable overhead (depending on your required throughput) associated with moving the data from the application to the USB device. If data were sent ‘byte at a time’ by an application, this would severely limit the overall throughput of the system as a whole.

Continuous data - smoothing the lumps

Data is received from USB to the PC by a polling method. The driver will request a certain amount of data from the USB scheduler. This is done in multiples of 64 bytes. The ‘bulk packet size’ on USB is a maximum of 64 bytes. The host controller will read data from the device until either (a) a packet shorter than 64 bytes is received or (b) the requested data length is reached. The device driver will request packet sizes between 64 Bytes and 4 KBytes. The size of the packet will affect the performance and is dependent on the data rate. For very high speed, the largest packet size is needed. For ‘real-time’ applications that are transferring audio data at 115200 Baud, for example, then the smallest packet is desirable; otherwise the device will be holding up 4k of data at a time. This can give the effect of ‘jerky’ data transfer if the USB request size is too large and the data rate too low (relatively).

Small amounts of data or end of buffer conditions

When transferring data to the PC, the device will send the data given one of the following conditions:

1. The buffer is full (64 bytes made up of 2 status bytes and 62 user bytes).

2. One of the RS232 status lines has changed (FT8U232 chip only). A change of level (high or low) on CTS / DSR / DCD or RI will cause it to pass back the current buffer even though it may be empty or have less than 64 bytes in it.

3. An event character had been enabled and was detected in the in-coming data stream.

4. A timer integral to the chip has timed out. There is a timer in both the FT8U232 and 245 chips that measures the time since data was last sent to the PC. The value of the timer is set to 16 milliseconds. Every time data is sent back to the PC the timer is reset. If it times-out then the chip will send back the 2 status bytes and any data that is held in the buffer.

From this it can be seen that small amounts of data (or the end of large amounts of data), will be subject to a 16-millisecond delay when transferring into the PC. This delay should be taken along with the delays associated with the USB request size as mentioned in the previous section. A worst case condition could occur where 62 bytes of data were received in 16 milliseconds. This would not cause a timeout but would send the 64 bytes (2 status + 62 user data) back to USB every 16 milliseconds. When the USBD system driver received the 64 bytes it would hold on to them and request another 'IN' transaction. This would be completed 16 milliseconds later and so on until USBD gets its 4 K of data. The overall time would be (4096 / 64) * 16 milliseconds = 1.024 seconds between data packets being received by an application. In order for the data to avoid arriving in 4K packets, it should be requested in smaller amounts. A short packet (< 64 bytes) will of course cause the data to pass from USBD back to our driver for use by the application.

- For application programmers it must be stressed that data should be sent or received using buffers and not individual characters.

Overcoming the Latency timer

To try to overcome the latency timer, one of the other conditions has to be met. That is:

1. The buffer is full (64 bytes made up of 2 status bytes and 62 user bytes).

2. One of the RS232 status lines has changed (FT8U232 chip only). A change of level (high or low) on CTS / DSR / DCD or RI will cause it to pass back the current buffer even though it may be empty or have less than 64 bytes in it.

3. An event character had been enabled and was detected in the in-coming data stream.

The most obvious way is to keep sending it data. In this way the data in continuously pushed through the chip and is not held waiting for a timeout.

Another method, that can be used by the FT8U232 chip only, is to change one of the modem status lines. This can be done by an external device or by the host PC its self. If an unused output line (DTR) is connected to one of the unused inputs (DSR), then it can be used to flush the buffer in the chip. If the DTR line is changed by the application program from low to high or high to low, this will cause a change on DSR and make it flush the buffer.

The last method is Event Characters. These can be used with either device. If the Event character is enabled and it is detected in the data stream, then the buffer is sent immediately. The event character is not stripped out of the data stream by the device or drivers. It is up to the application to deal with it. It may be turned on and off depending if you want to send large amounts of random data or small command sequences. The Event character does not work if it is the first character in the buffer. It needs to be the second or more. The reason for this was for applications that use the Internet, for example, will program up the event character as ‘$7E’. All the data is then sent and received in packets that have ‘$7E’ at the start and end of the packet. To maximise throughput and avoid a packet with only the starting ‘$7E’ in it, the event character does not trigger on the first position.

Flow Control

The FT8U245 chip uses handshaking as part of its design by proper use of the TXE# line. The FT8U232 chip can use RTS/CTS, DTR/DSR hardware or XON/XOFF software handshaking. It is highly recommended that handshaking is used.

There are 4 methods of flow control that can be programmed for the FT8U232 device.

1. None - this may result in data loss at high speeds

2. RTS/CTS - 2 wire handshake. The device will transmit if CTS is active and will drop RTS if it cannot receive any more.

3. DTR/DSR - 2 wire handshake. The device will transmit if DSR is active and will drop DTR if it cannot receive any more.

4. XON/XOFF - flow control is done by sending or receiving special characters. One is XON (transmit on) the other is XOFF (transmit off). They are individually programmable to any value.

Flow control is encouraged to be used because we are unable to ensure that we will always be scheduled. The chip can buffer up to 384 bytes of data. Windows can ‘starve’ the driver program of time if it is doing other things. The most obvious is moving an application around the screen with the mouse by grabbing its task bar. This will result in a lot of graphics activity and data loss will occur if receiving data at 115200 baud (as an example) with no handshaking. If the data rate is low or data loss is acceptable then flow control may be omitted.