USB BASICS FOR THE EXAR FAMILY OF USB UARTS

1.0 INTRODUCTION

Exar's family of USB UARTs includes the XR21V1410 / 2 / 4, single / dual / quad channel family as well as the XR21B1411 single channel UART. All of these devices are full speed USB peripherals. This application note discusses some basic background of USB to enable users of these devices to understand USB topology, power, bus protocol and a number of other subjects to better enable them to design with these devices as well as debug issues that might arise. For more information on these devices, refer to their respective data sheets available on the Exar website. For differences between the two families refer to the AN205 application note: Comparison of the XR21V1410 and the XR21B1411.

2.0 BACKGROUND

2.1 Advantages of USB

The advantages of USB are numerous. USB was designed as a low cost, serial interface solution with bus power provided from the USB host to support a wide range of peripheral devices. The original bus speeds for USB were low speed at 1.5 Mbps, followed by full speed at 12 Mbps, and then high speed at 480 Mbps. With the advent of the USB 3.0 specification, the super speed was defined at 4.8 Gbps. Maximum data throughput, i.e. the line rate minus overhead is approximately 384 Kbps, 9.728 Mbps, and 425.984 Mbps for low, full and high speed respectively. Note that this is the maximum data throughput and it can be adversely affected by a variety of factors, including software processing, other USB bandwidth utilization on the same bus, etc.

Another major advantage of USB is that it supports dynamic attachment and removal, which is a type of interface referred to as "plug and play". Following attachment of a USB peripheral device, the host and device communicate to automatically advance the externally visible device state from the attached state through powered, default, addressed and finally to the configured states. Additionally, all devices must conform to the suspend state in which a very low bus power consumption specification must be met. The device states that are visible to the USB and the host are defined in the table below (other states are internal to the USB device). Power conservation in the suspended state is another USB benefit.

TABLE 1: USB VISIBLE DEVICE STATES

ATTACHED	POWERED	DEFAULT	ADDRESS	CONFIGURED	SUSPENDED	STATE INFO
No	**-**	**-**	**-**	**-**	**-**	Device is not attached to the USB.
Yes	**No**	**-**	**-**	**-**	**-**	Device is attached to the USB, but is not powered.
Yes	**Yes**	**No**	**-**	**-**	**-**	Device is attached to the USB and powered, but has not been reset.
Yes	**Yes**	**Yes**	**No**	**-**	**-**	Device is attached to the USB and powered and has been reset, but has not been assigned a unique address. Device responds at the default address.
Yes	**Yes**	**Yes**	**Yes**	**No**	**-**	Device is attached to the USB, powered, has been reset, and a unique device address has been assigned. Device is not configured.
Yes	**Yes**	**Yes**	**Yes**	**Yes**	**No**	Device is attached to the USB, powered, has been reset, has a unique address, is configured, and is not suspended. The host may now use the function provided by the device.
Yes	**Yes**	**-**	**-**	**-**	**Yes**	Device is, at minimum, attached to the USB and is powered and has not seen bus activity for 3 ms. It may also have a unique address and be configured for use. However, because the device is suspended, the host may not use the device's function.

2.2 Development Challenges

USB also presents some challenges for developers. Two key technical hurdles are the overall complexity of the protocol and developmental changes across operating systems. The advantages of USB including the dynamic attachment, robust error checking mechanisms and versatility, come at the price of a complex protocol. An additional technical challenge is the evolving support for USB in various operating systems. As USB products and various Operating Sytems mature, changing support in the OS can create hazards for developers. A third non-technical challenge is the cost of membership in the USB organization as well as costs of compliance testing. Each USB device developer must have a membership in the USB organization, but OEMs using these chips may not need or desire to become USB members.

2.3 USB 2.0

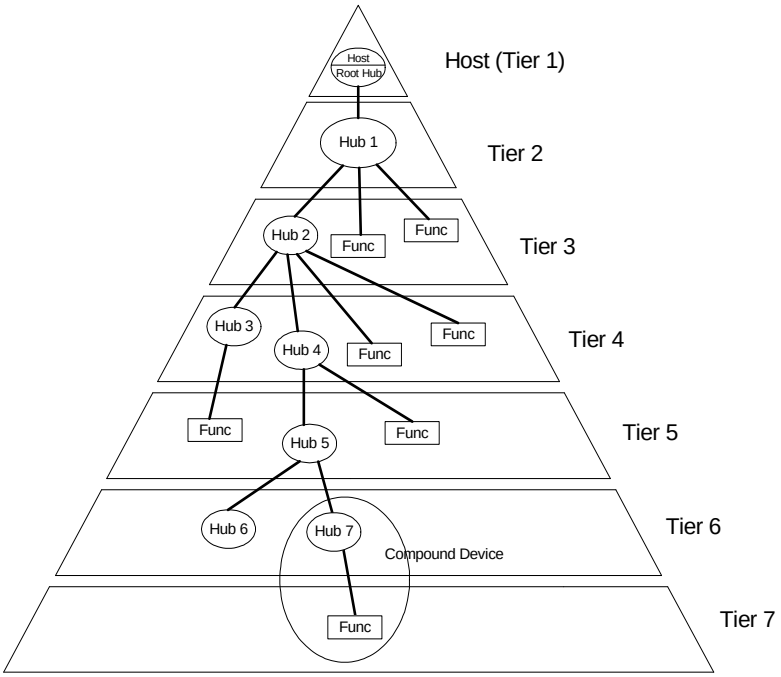
The USB 2.0 specification includes low, full and high speed device specifications. Compliance to USB 2.0 specification for peripheral devices does not necessarily indicate that the device is a high speed device, however a hub advertised as USB 2.0 compliant, must be high speed capable. The USB UARTs currently offered by Exar are all USB 2.0 compliant full speed devices.

3.0 USB ARCHITECTURE

3.1 BUS TOPOLOGY

USB devices fall into the category of hubs - which provide additional downstream attachment points, or functions - which provide a capability to the system. USB physical interconnection is a tiered star topology (see Figure 1). Starting with the host and "root hub" at tier 1, up to seven tiers with a maximum of 127 devices can be supported. Tier 2 through 6 may a have one or more hub devices in order to support communication to the next tier. A compound device (one which has both a hub and peripheral device functions) may not occupy tier 7.

FIGURE 1. USB BUS TOPOLOGY



The physical USB interconnection is accomplished for all USB 2.0 (up to high speed) devices via a simple 4-wire interface with bi-directional differential data (D+ and D-), power (VBUS) and ground. The VBUS power is nominally +5V. An "A-type" connector and mating plug are used for all host ports as well as downstream facing hub ports. A "B-type" connector and mating plug are used for all peripheral devices as well as the upstream facing port of a hub. Cable connections between host, hubs and devices can each be a maximum of 5 meters or ~16 feet. With the maximum of 7 tiers, cabling connections can be up to 30 meters or ~ 98 feet total.

3.2 Power

VBUS power from the host is limited. The power in USB is defined in terms of unit loads of 100 mA on the VBUS supply. Devices may be either bus-powered or self-powered. A bus-powered device draws all power from the VBUS power supplied by the USB cable. (Note, in this context, the term "device" refers to the USB interface component and associated electronic components.) A low power bus-powered function may draw a maximum of one unit load (100 mA) from the VBUS power. A high power bus-powered device may draw up to 5 unit loads (500 mA) from VBUS after being configured but may draw only 1 unit load before being configured. Devices that have an alternate source of power other than VBUS are termed self-powered. These devices may also draw up to one unit load of power from VBUS to maintain some level of functionality when the alternate power source is removed.

Hubs also fall into the categories of bus-powered and self-powered. Bus-powered hubs may also draw up to 1 unit load before and 5 unit loads after configuration. The downstream facing ports of a bus-powered hub will supply a maximum of 1 unit load. Self-powered hubs may also draw up to one unit load of VBUS power but must supply 5 unit loads to each downstream port. However, a self-powered hub whose alternate power is supplied from a battery may provide either 1 or 5 unit loads to each downstream facing port.

3.3 Protocol

3.3.1 USB Bus Enumeration

USB is a polled bus where all transactions are initiated by the USB host. All attached and removed USB devices are identified by a process termed "bus enumeration". An attached device is recognized by the host and its speed (low, full or high) is identified via a signaling mechanism using the D+/D- USB data pair. The attached device will initially utilize the default USB address of 0. Additionally, all USB devices are comprised of a number of independent endpoints which provide a terminus for communication flow between the host and device. The endpoint is a buffer that typically consists of a block of memory or registers which stores received data or contain data which is ready to transmit. Each endpoint is assigned a unique endpoint number determined at design time, however all devices must support the default control endpoint which is assigned number 0. Additionally, each endpoint is defined to support flow in one direction, labeled "Out", i.e. data from the host, or "In", i.e. data to the host. Communication from the host to each device endpoint uses a communication "pipe" which is established during enumeration. The pipe is a logical association between the host and the device. The combination of the device address, endpoint number and direction allows the host to uniquely reference each endpoint.

Once communication has been established with a device / function control endpoint, a unique (non-zero) address is assigned to that function and upon request from the host, the device supplies information regarding its attributes. Attributes are reported via a defined data structure called a descriptor. The devices vendor and product ID codes are part of the information provided to the host in descriptors. A unique vendor ID is assigned to the device manufacturer by the USB-IF organization and each manufacturer in turn may specify product ID codes to each of its products. Additionally, the maximum device power, bus or self-power mode and support for remote wake-up is reported to the host via descriptors.

3.3.1.1 Device identification by location

Each USB device that is enumerated by the host system is uniquely identified. To identify the devices, the host system uses the vendor and product IDs (VID and PID). Additionally, conforming to USB 2.0 specification, devices may or may not have a serial number. The XR21V1410 / 2 / 4 device family do not have serial numbers. When these devices are enumerated, they are uniquely identified by their VID / PID and their location in the USB tree, i.e. at which port of the root hub and which port of any external hubs they are connected to. If the device is moved to a different port, it will be identified as a new unique device. In this case the USB UART would be assigned a new COM port number.

3.3.1.2 Device identification by ID

Also by specification, if a device is serialized, it must have a unique serial number (for that VID / PID). Each XR21B1411 device has a unique serial number. When the device is enumerated, it is identified by this unique combination of VID, PID and Serial number. If the device is moved to a different port, it will be identified as the same device. In this case the USB UART would reatain the same COM port number.

3.3.2 Data Transfers

Once the enumeration is complete, the host and device are free to carry out communications via data transfers from the host to the device or vice versa. Both directions of transfers are initiated by the host. Four different types of transfers are defined (although low speed devices do not support bulk or isochronous transfers). These types are:

- Control Transfers: Used to configure a device at attach time and can be used for other device-specific purposes, for example device specific register read / write access as well as control of other pipes on the device. Control transfers consist of up to three distinct stages, a setup stage containing a request, a data stage if necessary to or from the host, and a status stage indicating the success of the transfer. USB has a number of standardized transactions that are implemented using control transfers. For example, the "Set Address" and "Get Descriptor" transactions are always utilized in the device enumeration procedure described above. The "Set Configuration" request is another standard transaction which is also used during device enumeration.
- Bulk Data Transfers: Capable of transfering relatively large quantities of data or bursty data. Bulk transfers do not have guaranteed timing, but can provide the fastest data transfer rates if the USB bus is not occupied by other activity.
- Interrupt Data Transfers: Used for timely but reliable delivery of data, for example, characters or coordinates with human-perceptible echo or feedback response characteristics. Interrupt transfers have a guaranteed maximum latency, i.e. time between transaction attempts. USB mice and keyboards typically use interrupt data transfers.
- Isochronous Data Transfers: Occupy a prenegotiated amount of USB bandwidth with a prenegotiated delivery latency. Isochronous transfers have guaranteed timing but do not have error correction capability. Isochronous data must be delivered at the rate received to maintain its timing and additionally may be sensitive to delivery delays. A typical use for isochronous transfers would be for streaming audio or video.

Note that Exar's USB UARTs all support Control Transfers via the Control Endpoint, Bulk Transfers for each UART channel via both Bulk In and Bulk Out Endpoints as well as an Interrupt In Endpoint. This conforms to the specifications for a Communications Device Class device. Note that the Interrupt Endpoint in Exar's USB UARTs is used only to carry a CDC-ACM compliant Interrupt Packet and cannot be used for data transfers. Additionally none of Exar's USB UARTs use Isochronous Data Transfers.

3.3.3 Device Classes

USB devices are divided into classes. The hub is a specially designated class of devices that has additional requirements in the USB specification. Other examples of classes of peripheral devices are human interface, also known as HID, printer, imaging, mass storage and communications. All of Exar's USB UART devices fall into the communications device class (CDC) of USB devices.

3.4 Host Controller

The USB host controller is responsible for numerous tasks, including the following:

- State Handling - The host controller is the element of the host that reports and manages all states.
- Frame and Microframe SOF token generation - The host controller provides start of frame (SOF) tokens at 1 ms intervals when operating with full speed devices and 125 us (micro-frame) intervals when operating with high speed devices. Timing of these tokens from the host is guaranteed within limits specified in the USB 2.0 specification. The SOF token is the first transmission in the frame or micro-frame transmission.
- Error handling - The host controller detects the following error conditions:
  - Timeout conditions after a host transmission, for example when the addressed endpoint is unresponsive.
  - Data errors, including received packets with CRC error.
  - Protocol errors, for example invalid handshake PID or bit stuffing error.

- Suspend mode and remote wake-up - To place the USB bus in a global suspend state, the host controller stops all USB traffic including SOF generation. In this state, the host controller may be enabled by the host system to respond to remote wake-up events.
- Root hub - The root hub provides the connection between the Host Controller and one or more USB ports.

3.5 USB Device Responsibilities

Many of the USB device responsibilities are a mirror image of the host responsibilities, including:

- Detecting communications to the device - the device must detect and respond to its assigned USB address.
- Respond to standard requests - USB standard requests, as well as possibly class requests and vendor specific requests.
- Error checking - Similar to the host, a device performs error checking calculations.
- Power management - A device must publish to the host, its power requirements for VBUS power drawn from the host.
- Exchange Data with the host - This is the primary purpose of the USB device. All other functionality exists to support the device capability to exchange data with the host. A device receives data from the host and can only send data in response to a request from the host (although a Super-speed device has a packet to elicit the host request).

4.0 SOFTWARE IMPLEMENTATION

4.1 Software stack

USB protocol is supported in numerous operating systems, including all modern versions of Windows, Mac OS X, and Linux. A typical implementation for USB communications uses a layered driver model where each driver in a series or "stack" performs a portion of the communication. A USB system is the component layer of the host that uses the Host Controller to manage data transfers between the host and USB devices. The USB systems have three basic components:

- Host Software
- USB Driver
- Host Controller Driver

4.2 Software drivers

The host controller driver or HCD is at the lowest tier of the software stack and provides an abstraction to the host controller hardware. This abstraction "hides" the low level details of the hardware. Below the host controller hardware is the physical USB including all attached devices. The HCD client is the Universal Serial Bus Driver (USBD).

The USBD provides a collection of mechanisms that operating systems components, (typically device drivers), use to access USB devices. The USB driver is operating system specific. The USBD may use mechanisms provided by the OS environment as needed and may augment them when necessary. The USBD provides data transfer mechanisms in the form of I/O Request Packets (IRPs), which consist of a request to transport data across a specific pipe.

Device manufacturers (or operating system vendors) must provide the necessary device driver software and client interface software to convert their device from the physical implementation to a USB-compliant software implementation (or virtual device). The Host Controller provides service based on parameters provided by the Host Software, (e.g. a test application GUI), when the configuration request is made.

4.2.1 CDC-ACM driver

Because Exar’s USB UARTs are fully compliant to the Communications Device Class (CDC) standard, they will work with a standard CDC-ACM driver (CDC - Abstract Control Model). Both Windows and Linux

operating systems supply a default CDC-ACM driver that can be used with Exar's USB UART devices. In Windows this driver file name is usbser.sys. In Linux, this driver file name is cdc-acm. CDC-ACM drivers can also be supplied by third party vendors.

The CDC-ACM driver will specifically support a subset of the option device class specific commands that are utilized in Exar's UARTs, i.e. the SET_LINE_CODING and GET_LINE_CODING, SET_CONTROL_LINE_STATE and SEND_BREAK class specific commands. These commands are described below:

- SET_LINE_CODING - Configures data rate, number of character and stop bits, and parity
- GET_LINE_CODING - Queries data rate, number of character and stop bits, and parity
- SET_CONTROL_LINE_STATE - Sets or clears UART signals RTS and DTR
- SEND_BREAK - Requests to assert break condition

4.2.2 Custom Drivers

The "default" or CDC-ACM driver has limited capabilities to control specific devices. It utilizes the class specific requests described above to set or query basic device performance. However this generic driver does not have any knowledge about specific device memory mapped registers. Because of this, device manufacturers can create an alternate, or custom driver that is capable of accessing the device specific register sets. In Windows this driver file name for all of Exar's USB UARTs is xrusbser.sys. In Linux, for the XR21V1410 / 2/ 4 family this file is vizzini and for the the XR21B1411 device this file is mongo. These drivers allow application software to utilize the device registers to configure or utilize other specific features in the devices. Exar's custom drivers utilize vendor specific USB messages XR_SET_REG and XR_GET_REG. These drivers can be downloaded from Exar's website.

Note that in Linux, a specific set of instructions to load the custom driver are given with the Exar driver in a readme file.

4.2.3 INF Files

When using a USB driver in Windows, the developer must also supply an information or .inf file. The .inf file is a text file that specifies the files that need to be present or downloaded for your component to run. For the default CDC-ACM driver this INF file specifies the usbser.sys file as the appropriate driver.

For cusom drivers, the INF file specifies the appropriate driver (xrusbser.sys) based upon specified VID / PID combinations. INF files provide the ability to create customized software installation instructions, which include registry entries and destination directories. Additionally, by pointing to the URLs of files to download, an .inf file may enable the operating system of an internet connected host system to download files and install them.

4.3 Host Controller Interface

The Host Controller Interface or HCI is a register-level interface that enables a host controller for USB hardware to communicate with a host controller driver (HCD) in software. There are a number HCI standards which are specified below. Note that not all of the HCI standards below are supported on all root hub ports for a given host controller.

4.3.1 OHCI

The Open Host Controller Interface or OHCI is an open standard. The OHCI standard for USB supports USB 1.1 (full and low speeds) only.

4.3.2 UHCI

Universal Host Controller Interface (UHCI) was created by Intel for USB 1.0 (full and low speeds). It is incompatible with OHCI. Intel and VIA controllers generally use UHCI, while other vendors (for example AMD) use OHCI.

4.3.3 EHCI

Enhanced Host Controller Interface (EHCI) is a high-speed controller standard that is publicly specified. EHCI only provides high-speed USB functions. It relies on a "companion controller", either OHCI or UHCI, to handle full- and low-speed devices. Motherboards and PCI Cards that provide high-speed ports thus have two controllers, one handling high-speed devices and the other handling low- and full-speed devices. It is not uncommon to find UHCI, OHCI and EHCI all co-existing in a standard PC, with a UHCI driver providing low- and full-speed functions on the (Intel chipset) motherboard, an OHCI driver providing low- and full-speed functions for the USB ports on an add-in (NEC chipset) PCI expansion card, and an EHCI driver providing high-speed functions for the USB ports on that expansion card.

4.4 WHQL Certification

WHQL Testing (Windows Logo Program) is Microsoft's testing process which involves running a series of tests on third-party hardware or software, and then submitting the log files from these tests to Microsoft for review. The procedure may also include Microsoft running their own tests on a wide range of equipment, like different hardware and different Microsoft Windows editions. Products that pass the WHQL tests get to use a "Certified for Windows" logotype, which certifies that the hardware or software has had some share of testing by Microsoft to ensure compatibility. For device drivers passing the WHQL tests, Microsoft creates a digitally signed certification file. Microsoft 64 bit operating systems will not properly load device drivers without this certification.

Exar provides WHQL certified custom drivers for all of its USB UART devices.

5.0 USB COMPLIANCE

To complement the USB specification and enable measurement of compliance in real products, the USB-IF has instituted a Compliance Program that provides reasonable measures of acceptability. Products that pass this level of acceptability are added to the Integrators List and have the right to license the USB-IF Logo.

All of Exar’s USB UARTs (as well as all evaluation board designs) have been certified and are listed on the USB-IF Integrators List. Note that although a silicon vendor, e.g. Exar, may certify a specific device, an OEM utilizing that device must also certify its product with the device embedded in order to be USB certified. This is necessary since many of the tests are system level tests, for example USB power consumption.

6.0 TECHNICAL SUPPORT

Technical questions about this application note, the USB UART devices, evaluation boards, or software drivers should be e-mailed to uarttechsupport@exar.com.

A.1 Appendix A1 - USB Defintions

- Downstream - The direction of data flow from the host or away from the host. A downstream port is the port on a hub electrically farthest from the host that generates downstream data traffic from the hub. Downstream ports receive upstream data traffic.
- Endpoint - A uniquely addressable portion of a USB device that is the source or sink of information in a communication flow between the host and device. See also endpoint address.
- I/O Request Packet (IRP) - An identifiable request by a software client to move data between itself (on the host) and an endpoint of a device in an appropriate direction.
- Pipe - A logical abstraction representing the association between an endpoint on a device and software on the host. A pipe has several attributes; for example, a pipe may transfer data as streams (stream pipe) or messages (message pipe). USB Function - A USB device that provides a capability to the host, such as an ISDN connection, a digital microphone, or speakers.
- Root Hub - A USB hub directly attached to the Host Controller. This hub (tier 1) is attached to the host.
- Transaction - The delivery of service to an endpoint; consists of a token packet, optional data packet, and optional handshake packet. Specific packets are allowed/required based on the transaction type.
- Transfer - One or more bus transactions to move information between a software client and its function.
- Upstream - The direction of data flow towards the host. An upstream port is the port on a device electrically closest to the host that generates upstream data traffic from the hub. Upstream ports receive downstream data traffic.
- USB Device - A logical or physical entity that performs a function. The actual entity described depends on the context of the reference. At the lowest level, device may refer to a single hardware component, as in a memory device. At a higher level, it may refer to a collection of hardware components that perform a particular function, such as a USB interface device. At an even higher level, device may refer to the function performed by an entity attached to the USB; for example, a data/FAX modem device. Devices may be physical, electrical, addressable, and logical. When used as a non-specific reference, a USB device is either a hub or a function.
- USB Function - A USB device that provides a capability to the host, such as an ISDN connection, a digital microphone, or speakers.

A.2 Appendix A2 - References and Related Documents

1.) Universal Serial Bus Specification, Rev. 2.0 Apr. 27, 2000

2.) USB Complete Fourth Edition by Jan Axelson

3.) Universal Serial Bus Class Definitions for Communication Devices, Version 1.1 January 19, 1999

4.) Exar Application Note #205 - Comparison of the XR21V1410 and the XR21B1411

5.) XR21V1410, XR21V1412, XR21V1414 Data Sheets, Exar Corporation

6.) XR21B1411 Data Sheet, Exar Corporation

NOTICE

EXAR Corporation reserves the right to make changes to the products contained in this publication in order to improve design, performance or reliability. EXAR Corporation assumes no responsibility for the use of any circuits described herein, conveys no license under any patent or other right, and makes no representation that the circuits are free of patent infringement. Charts and schedules contained here in are only for illustration purposes and may vary depending upon a user's specific application. While the information in this publication has been carefully checked; no responsibility, however, is assumed for inaccuracies.

EXAR Corporation does not recommend the use of any of its products in life support applications where the failure or malfunction of the product can reasonably be expected to cause failure of the life support system or to significantly affect its safety or effectiveness. Products are not authorized for use in such applications unless EXAR Corporation receives, in writing, assurances to its satisfaction that: (a) the risk of injury or damage has been minimized; (b) the user assumes all such risks; (c) potential liability of EXAR Corporation is adequately protected under the circumstances.

Datasheet March 2011.

Send your UART technical inquiry with technical details to hotline: uarttechsupport@exar.com.