Moto Development Group
85 Second Street
San Francisco, CA 94105

USB to Serial

Reference Design
MO1002

Rev 3.0 04/11/01

Product Overview

The MO1002 is a HID compliant, bi-directional USB to Universal Asynchronous Serial Receiver/Transmitter (UART) solution. Devices that currently employ asynchronous serial communications can be connected to a USB host without a major redesign of the hardware. While USB to UART solutions currently exist, the MO1002 is lower cost and uses proven driver technology from Microsoft®.

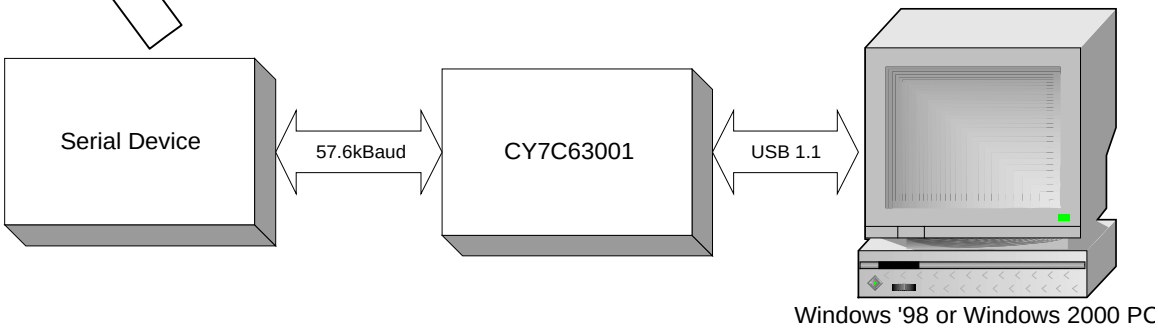
For more information contact Ben Knapp at MOTO at (415) 281-4800 or visit our web site at www.moto.com/usb.

Features

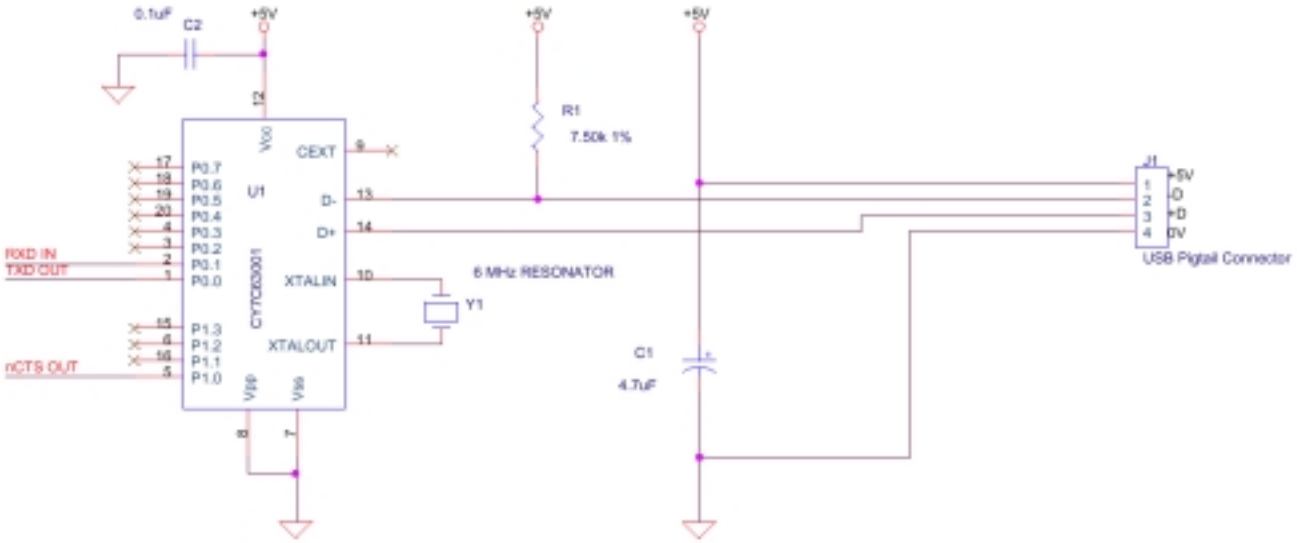
- Rates From 2400 To 57.6 kbps
- USB 1.1 Compliant
- All Necessary Firmware Included
- Small Footprint (24 lead QSOP Package available)
- Uses USB Human Interface Device driver which is not operating system-specific

Benefits

- Lowest Cost Solution Available
- Xon/Xoff Software Flow Control
- Hardware Flow Control



Schematic



Bill of Materials

Item	Quantity	Reference	Part	Rating	Tol	Footprint	Supplier	Partnumber
1	1	C1	4.7uF	16V	_20%	CAP4X7	Panasonic	ECE-A1CKA100
2	2	C2	0.1uF	50V	_20%	CAP	Philips	A104Z15Z5UFVVWN
3	1	R1	7.5K	_1/8W	_1%	RES		
4	1	U1	CY7C63001A			SOIC or DIP or QSOP	Cypress	CY7C63001A-SC or CY7C63001A-PC or CY7C63101A-QC
5	1	Y1	6MHz			CRES	Murata	

Developer Notes

FEATURES

- Comes out of reset at default baud rate of 9600
- Six discrete baud rates can be set using host software (2400, 4800, 9600, 19200, 38400, 57600)
- Hardware flow control. The device will signal when its receive buffer (60 bytes) is full.

LIMITATIONS

- Half duplex only. Device cannot transmit and receive at the same time. If the host directs the device to transmit, incoming (received serial data) will be ignored.

UNCERTAINTIES

- Low power suspend. MOTO found it necessary to rework the low power suspend functionality. It hasn't been tested recently, but it ought to work.
- If the device is receiving a serial character and the host starts sending control transfers (to either adjust the baud rate or to cause characters to be transmitted) the character being received may be abandoned.

BUILDING

Firmware

The firmware is assembled using Cypress Semiconductor's CYASM.EXE program. The latest version of the assembler is available free from Cypress at www.cypress.com.

In order to get started quickly, you'll want to contact Cypress about their CY3649 Hi-Lo Programmer for the CY7C63xxx series of devices.

To assemble the source, type CYASM HIDUART. HIDUART.ASM is the main source file which includes all the others. As released by MOTO, the source generates no errors or warnings. The output file is in HEX format, and is named HIDUART.HEX.

Software

The HIDUART terminal program can be built using Visual C++ 5.0. Other compilers have not been tested

USAGE OF VENDOR ID, PRODUCT ID, AND DEVICE ID

The source uses MOTO's vendor ID (0x0C04) and a product ID of 0x0100.

These Vendor and Product ID values belong to MOTO Development Group. If you use MOTO's vendor ID you may use PID 0x0100 only AND you may NOT use version (aka device ID) 0100 or lower. If you choose this path your product must be differentiated by its device ID only. Your application must test for vendor then product then device ID before it attaches to the HID device. See the example HIDUART.EXE source. If you want your own product ID MOTO may be willing to sell one of ours. Otherwise, USB.ORG is the sole broker of vendor IDs.

Further, all of the above is true only insofar as it is not in contravention of MOTO's obligations as a member of the USB Implementors Forum.

FIRMWARE THEORY OF OPERATION

Human Interface Device (HID) Class of USB Devices

Developing USB device drivers is non-trivial. Using the existing Microsoft HID class driver eliminates the driver development work, but it puts requirements on the firmware to "speak HID." Fortunately, Cypress' "firmware frameworks" (for the CY7C63x0x parts) implements a HID interface. MOTO added the UART functionality to the frameworks.

Serial Transmit Mechanism
The host sends a HID Set Report of type RID_TRANSMIT. The data in this report is pushed out the TX pin of the device as asynchronous serial data at the set bit rate. SerialTransmitByte does the work in this case.

Serial Receive Mechanism
SerialReceiveByte is the routine that assembles incoming bytes. It is called in the event of a negative edge triggered GPIO interrupt. The start bit of the serial byte causes this interrupt. The routine maintains control until the whole byte is collected. Each bit is sampled only once, at roughly the center of the bit. There is no parity checking. A return from interrupt is executed at the completion of this routine.

The value of POLLING_INTERVAL controls how often the host PC sends down "INTERRUPT IN" tokens. The firmware can only send data up to the host in response to an INTERRUPT IN token, so this is a direct control of the UART receive capability of the device. If the polling is too infrequent your UART receive buffer will overflow because it can't send data up to the host quickly enough. If it is too frequent you may cause problems with other USB devices.

Now, most HID devices put 10 here, to indicate "poll me every 10 milliseconds" for new data. In fact, when you put 10 or values near 10 here Windows will poll the device every 8 milliseconds. Further, you can put values as low as 1 millisecond in here and it still works in our limited testing. Devices are not supposed to use numbers less than 10 because they chew up USB bandwidth, but using ‘1’ does successfully transfer data at a steady clip of 7000 bytes / second (56000 bps). In that case the device can receive data at 57600 baud and will rarely issue a flow control signal.

Baud Rate
The baud rate comes up at a default rate as defined in the file ram.inc, in the constant BAUD_BITDELAY_DEFAULT. This value lies in the baud select table in hiduart.asm at the label “baud_table.”

A special Set Report is defined to change the baud rate to one of the values in this table.

Timing the signals with an oscilloscope it can be seen that the bit durations are slightly wrong. The firmware was written to operate at 57600 bps using very simple UART code. From there, the best available values for bit delay times were found to create the other baud rates. There is no doubt that the code could be re-written to achieve more accurate results at any particular baud rate at the expense of being able to do many baud rates well. Further, all baud rates could be done more precisely with separate code to handle each rate. Fortunately the performance as the design exists appears to be reliable.

HIDUART Asynchronous Serial Baud Rate Generation (TX)
Times are in microseconds

		Actual		
Rate	Ideal	start bit	high data bit	low data bit
2400	416.67	412.8	414.2	413.5
4800	208.33	204.5	205.8	205.8
9600	104.17	101.4	102.7	102.4
19200	52.08	49.8	51.05	50.9
38400	26.04	24.02	25.33	25.08
57600	17.36	16.08	17.4	17.16