The Bit-Banger
USB to $I^2C$ bridge

Circuit Cellar Design 2K Contest
Entry #D2K127
Overview

The BitBanger board and device driver provide a simple method to quickly interface a wide variety of I²C components to a Windows 98 P.C. through a USB port.

The design consists of the USB to I²C bridge hardware (BitBanger Board), and a Windows 98 compatible WDM device driver (BitBang.Sys).

An example Visual Basic application (BitBanger.Exe) is also provided to demonstrate how to communicate with the driver and hardware. The I²C functionality is demonstrated with a simple circuit which provides 16 bits of digital I/O. The 16 bits can be read and written from within the BitBanger.exe example application.

There are many components available that incorporate I²C control interfaces, including eeproms, rams, DACs, ADC’s, multiplexers, digital I/O, USB, etc.

Almost any combination of the currently available I²C components could be controlled by the BitBanger.

Usage

The BitBanger is primarily intended for use by technical people such as hardware engineers, software engineers, and electronics hobbyist.
**Title**

I2C to USB

**Notes:**

1. Connect +3.3VA and +3.3V at C4.
2. Connect AGND and DGND at C4.
Components

The heart of the circuit is a Philips 87LPC764 micro-controller. This component is an 8051 derivative featuring 4KB of one-time programmable ROM, 64 bytes of RAM, and an assortment of on-chip resources (timers, UART, \( \text{I}^2\text{C} \), reset, etc.). All of the above features are shoe-horned into a 20 pin package. Because the part implements an 8051 core, a wide variety of development tools are available. The Philips 87LPC764 micro-controller has a wide operating voltage range (2.7V to 6V) which allows it to operate from \( V_{\text{usb}} \) (note: Section 7.2.2 of the USB specification shows this voltage can be as low as 4.375V), far below the minimum voltage required by many 5V parts. The datasheet for this device is available at http://www.phillipssemiconductor.com/pip/P87LPC764FD.

The connection to the host computer is achieved with a Philips PDIUSBD11 USB interface. This component features an \( \text{I}^2\text{C} \) control interface requiring only three signals from the micro-controller. The device supports full speed USB connections (12Mbps), includes memory buffers for the USB transfers, and has an integrated SIE (serial interface engine) to handle the significant transaction protocol and speed requirements of USB. A SoftConnect (Philips Trademark) feature provides support for disconnection/connection signaling to the USB via software control, and the programmable clock output allows us to keep the crystal count (for the design) down to one (12Mhz). The datasheet for the PDIUSBD11 is available at http://www-us.semiconductors.com/usb/products/interface/pdiusbd11/.

The circuit includes a Micrel 2920A-3.3 voltage regulator for supplying +3.3V to the PDIUSBD11 (USB interface I.C.) when the circuit is operating from USB power or when +5V is supplied via the \( \text{I}^2\text{C} \) connector. The regulator has a low dropout voltage of only 370mV at 250mA, burns a mere 140uA of quiescent current, and can source 400mA. The datasheet for the 2920A-3.3 is available at http://www.micrel.com/product-info/products/mic2920a.html.

Power Options

The circuitry supports five jumper-configurable, power options:

1. Powered by USB, isolated from \( \text{I}^2\text{C} \) power (\( \text{I}^2\text{C} \) power used only for \( \text{I}^2\text{C} \) signal pull-up resistors).
2. Powered by USB, sourcing \( V_{\text{usb}} \) to \( \text{I}^2\text{C} \) bus.
3. Powered by USB, sourcing +3.3V to \( \text{I}^2\text{C} \) bus.
4. Powered by +5V supplied via \( \text{I}^2\text{C} \) bus connector, isolated from USB bus power.
5. Powered by +3.3V supplied via \( \text{I}^2\text{C} \) bus connector, isolated from USB bus power.
Initialization

At power up, the micro-controller initializes the USB and I\(^2\)C interfaces.

The processor is clocked by the PRGCLK (programmable clock) signal which is output from the PDIUSBD11. This signal is derived from a 48Mhz clock passed through a programmable divider and defaults to 4Mhz at power up. The divide value for the clock is changed from the default of 12 \((48\text{Mhz}/12 = 4\text{Mhz})\) to 3 \((48\text{Mhz}/3 = 16\text{Mhz})\) effectively shifting the processor into high gear.

Enumeration

After initialization, the firmware enters a forever loop where it waits for events to occur.

The first event (actually a whole series of events) to occur is USB enumeration.

The USB bus is designed to be plug-and-play compatible, requiring a method for detecting and initializing new devices, this method is called “enumeration”.

There are several situations that can result in enumeration (power up, plug-in, reset, etc.) but in this design they are all handled (from a software perspective) as if the device were just plugged in to the USB bus and the computer just detected its presence.

When the host computer detects our device “plug-in” it will initiate a series of USB transactions with the intent of discovering what our device is. The information it obtains from our device during enumeration allows the host computer to determine what drivers it should load to support our device and what the communication features of our device are (things like maximum packet sizes, power requirements, etc.). During the enumeration process, the host assigns the device a unique address (from 1 to 127), all further communications between the host and the device will use the assigned address.

The USB enumeration process is detailed in chapter 9 of the USB specification which is available at [http://www.usb.org/developers/data/usb_20.zip](http://www.usb.org/developers/data/usb_20.zip)
Operation

After enumeration, the device is ready to perform its intended function as a USB to I²C communications bridge.

The host will send I²C transaction requests to the BitBanger via the USB connection and the device driver calls ReadI2C, WriteI2C, and GetStatus.

The I²C transaction requests specify the type of transfer (no address, 8 bit address, or 16 bit address), the device ID, whether a Read or Write operation is to be performed, and how many bytes to transfer.

The PDIUSBD11 receives the specified I²C transaction requests (packaged up for USB) from the host and signals the processor via the nIRQ signal. The processor then communicates with the PDIUSBD11 via the I²C bus to service the interrupt and read the received requests from the PDIUSBD11’s buffer(s).

The processor then executes the specified I²C transactions, issuing the corresponding Read or Write requests over the I²C bus to the external device(s) attached to J1. In the event of a Read transaction, the read data is returned to the host.

Status information for the last executed transaction can be obtained from the Bit-Banger detailing the success or failure of the last executed transaction, the number of bytes transferred, and whether the device is currently busy.

In addition to the I²C bridge function, the Bit-Banger provides for Read and Write access of any register on the 87LPC764 processor via the ReadReg and WriteReg device driver calls.
A Brief Introduction to I²C

I²C is a Master/Slave serial protocol utilizing only two signals, SCL and SDA.

All devices attached to an I²C bus utilize ONLY open collector outputs to drive the signals and the signals are passively pulled high (resistors) whenever the bus is idle.

The SCL signal is the clock provided by the Master in a transaction, and the SDA signal is for the bi-directional data.

Transactions consist of a START sequence, an ID_ADR_MODE sequence, the DATA sequence(s) (each byte followed by an ACK/NAK, except the last), and are terminated by a STOP sequence. The ID_ADR_MODE sequence transfers a byte on the bus containing three pieces of information, a four bit device identification (specific to particular I²C components), a three bit address field (if you have two components with the same device ID, this is where you specify which one), and one mode bit (Read or Write). The ACK/NAK sequence allows the Master to signal that a data byte was received (Read) or the Slave to signal that a data byte was received (Write).

Some devices extend the basic protocol by utilizing a modified (un-terminated) write sequence chained together with another Read or Write sequence. This is common for devices with extended addressing requirements such as eeproms, rams, etc.

In the case of an eeprom, the modified write sequence allows the Master to specify which address it wishes to read from or write to, along with the data. The extended transaction sequence then consists of a START sequence, an ID_ADR_MODE sequence (Mode = Write), the DATA sequence(s) specifying the starting address for the operation (each byte followed by an ACK/NAK, except the last), a second ID_ADR_MODE sequence (Mode = Read or Write), a second DATA sequence(s) for the actual transaction data (each byte followed by an ACK/NAK, except the last), and is terminated by a STOP sequence.

There are many components available that with I²C control interfaces, including eeproms, rams, DACs, ADC’s, multiplexers, digital I/O, USB, etc.

In this design the I²C bus is enhanced by the inclusion of a third signal “nIRQ”. This signal is driven the same way (open collector, passive pull-ups) as the SCL and SDA signals, and provides a mechanism for event signaling from Slave to Master. Without the nIRQ line, a Master would have to continually poll any slaves capable of reporting events (signaling devices). With the nIRQ line, a system with only one signaling device can run in pure interrupt mode, and a system with multiple signaling devices can turn off polling whenever the interrupt line is inactive (this can greatly reduce the processor load).

More information on the I²C bus can be found at http://www-us.semiconductors.philips.com/i2c/.
Source Code (firmware)

The source code files for the firmware are contained in the file “firmware.zip”.

Source Code (Windows WDM device driver)

The source code files for the Windows WDM device driver are contained in the file “driver.zip”.
# Bill of Materials

<table>
<thead>
<tr>
<th>Item</th>
<th>Quantity</th>
<th>Reference</th>
<th>Part</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>CX1,CX2,CX3,CX4</td>
<td>0.1uF</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>C1,C2</td>
<td>33pF</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>C3,C4,C5</td>
<td>10uF</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>D1</td>
<td>LED, GREEN</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>JP1</td>
<td>JUMPER 3 PIN</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>JP2</td>
<td>JUMPER 2 PIN</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>J1</td>
<td>CON5 MALE</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>J2</td>
<td>CONN IDC 14</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>J3</td>
<td>USB Type B</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>R1,R2,R3</td>
<td>3.3K</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>R4</td>
<td>470</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>R5,R6</td>
<td>22</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>R7</td>
<td>10K</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>R8</td>
<td>100K</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>R9</td>
<td>4.7K</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>TP2,TP1, TP3</td>
<td>TESTPOINT</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>U1</td>
<td>87LPC764</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>U2</td>
<td>PDIUSBD11</td>
</tr>
<tr>
<td>19</td>
<td>1</td>
<td>U3</td>
<td>MIC2920A-3.3BS, 400mA</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>Y1</td>
<td>12MHz</td>
</tr>
</tbody>
</table>
Example Application (I²C I/O)
The example application “BitBang.exe” is a simple demonstration of how to access an I²C device using the Bit-Banger. The application demonstrates how to detect the presence or absence of the Bit-Banger device, obtain a handle, and execute I2C transactions calls.

Example Source Code (Visual Basic)
The source code files for the Example Visual Basic application are contained in the file exmplapp.zip.