Universal Serial Bus Host Interface on an FPGA

For many years, designers have yearned for a general-purpose, high-performance serial communication protocol. The RS-232 and its derivatives have long since been eclipsed by the requirements for speed and integrity of data. The emerging Universal Serial Bus (USB) specification may well prove to be the answer to the needs of system designers for now and the future.

USB Background

The USB was developed by a consortium of companies, including IBM, Microsoft, Intel, and others. These companies saw a need for a robust link between PCs and telephones. They also sought to make PC peripheral ports more expandable to support more devices outside the card cage. Concomitant with the expandability requirement are support for a range of transmission rates for different devices, maintenance of data integrity, and automatic PC reconfigurability. Overall, the goal of the USB is to provide a low-cost, flexible, easy-to-use bus that can support PC expansion at real-time audio/video data rates.

USB Specification

The USB is defined as an industry standard, which can be implemented by any company that produces a device to communicate with other USB devices. The standard defines communication protocols, transactions, electrical characteristics, and bus management.

Some of the applications for the USB are shown in Table 1, which categorizes them according to performance requirements. The USB consists of a host and its devices, which are connected in a hierarchical topology, as shown in Figure 1.

Table 1 • USB Applications

<table>
<thead>
<tr>
<th>Performance</th>
<th>Applications</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low speed</td>
<td>Keyboard, mouse</td>
<td>Lower cost</td>
</tr>
<tr>
<td></td>
<td>Stylus</td>
<td>Hot plug-unplug</td>
</tr>
<tr>
<td></td>
<td>Game peripherals</td>
<td>Ease of use</td>
</tr>
<tr>
<td></td>
<td>Virtual reality peripherals</td>
<td>Multiple peripherals</td>
</tr>
<tr>
<td></td>
<td>Monitor configuration</td>
<td></td>
</tr>
<tr>
<td>Medium speed</td>
<td>ISDN</td>
<td>Low cost</td>
</tr>
<tr>
<td>Phone, audio, compressed video</td>
<td>POTS</td>
<td>Ease of use</td>
</tr>
<tr>
<td></td>
<td>Audio</td>
<td>Guaranteed latency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Guaranteed bandwidth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dynamic attach-detach</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Multiple devices</td>
</tr>
<tr>
<td>High speed</td>
<td>Video</td>
<td>High bandwidth</td>
</tr>
<tr>
<td>Video, disk</td>
<td>Disk</td>
<td>Guaranteed latency</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ease of use</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The USB protocol supports three types of communication packets: token, data, and handshake. Depending on the purpose of the communication, a transfer may consist of all three packet types, a token and handshake, or a token alone. A three-packet transfer is for normal data transfers with handshaking. A simplified view of the fields in each packet in a three-packet transfer is shown in Figure 2. The fields of a single-stage or token-only packet are shown in Figure 3. (Not shown are the 1-byte synchronizer fields that head each packet and are filtered out by the data synchronizer hardware.)

**Figure 2 • Three-Stage Transfer Packet Fields**

The low granularity of the logic modules on the 3200DX devices make them good candidates for efficient design synthesis because the synthesizer can map library components directly to the logic modules.

The design description presents a somewhat simplified view of the USB specification so as to focus on the role of the FPGA in the host system. We do not discuss operating system issues such as how to process erroneous or incomplete data transmissions. The status for such occurrences are encoded according to the requirements and passed to the operating system running on the CPU.

**USB Host System Architecture**

The architecture of the USB host interface is shown in Figure 4. There are data paths between the bus and the host supported by a FIFO attached to bidirectional data paths. On the host system side, the FIFO sends received data to the main memory using DMA. (The DMA controller is system specific and will be left as a block.) In data transmissions, the host loads data to the FIFO from main memory.

The host establishes and maintains a connection table that contains target data for token formation and time stamps. The table contains all the device-specific information required to make transfers between the bus and the host. The operating system initiates transfers by loading the Packet Type register with the packet type code and with the connection table starting address to point to the PID code and to the target device description in the connection table.

Transactions requiring acknowledgment are terminated by an overflow of the time-out counter. The Status register provides feedback to the host operating system as to the results of bus transfers.

On the USB side of the data path, a serializer and a Shift register convert data between serial and parallel formats. Data is pushed or popped from the FIFO according to the direction of the data. The CRC is used to append a code to outgoing packets and to compare against the codes of incoming data.

**Figure 3 • Start-of-Frame Packet Fields**

A USB Interface on an FPGA

Developers of interfaces to buses, such as the USB, specified without a carrier clock have many asynchronous events to contend with. For example, the USB host must phase lock to asynchronous data transmitted from any one of several possible frequencies within eight transmitted bits. Designing a system that can asynchronously detect a valid data stream and phase lock to it throughout the transmission is a delicate task. Debugging such systems can only be done in hardware at actual clock and data rates. The debugging usually requires several iterations of the design, making an ASIC implementation impractical.

Using an FPGA to prototype the design for system verification is the safest and lowest-cost path to validating the design. For the design example described here, we use a member of the Actel 3200DX family of devices. These devices have the resources and the performance to implement a working system.
All data transfers are NRZI coded and decoded. NRZI encoding represents a logical 1 as no change in level and a logical 0 as a change in level. The NRZI coding state graph is shown in Figure 5.

Transmissions are also bit stuffed to retime the receiver logic. Stuffed bits must be inserted by the transmitter and discarded by the receiver. The bit insertion state graph is shown in Figure 6. Both the NRZI and Bit-Stuffer state machines are alerted to the presence of data on the next clock edge by the Data Flow Control state machine.

At the edge of the USB and the interface is the synchronizing logic. Eight bits of synchronization data are sent by the transmitter prior to each packet. The synchronization data are discarded by the receiver and is not shown in the packet field diagrams.
The transmit side of the system described multiplexes outgoing synchronization and communication data so as not to NRZI code or bit-stuff code the synchronization byte. Data transfer frequency is selected through the clock multiplexer according to the target frequency. On the receive side, there is a phase lock loop to detect a synchronization clock and align incoming data to the local clock.

**Operation**
The state machine assembles packets under the direction of the operating system, which determines the schedule composed of slots, frames, and superframes. Frames consist of a start-of-frame (SOF) token and successive transfers. The Assembly state machine is keyed to the beginning of a frame by the operating system and inserts an SOF token at that point. A superframe is made up of several frames, each of which has its own SOF token.

Packets can consist of tokens only when all transmitted data comes from the connection table. Other packets include a token and data packet, with the latter coming from the FIFO and being appended to the token by the state machines.

There are state machines to control the packet and data transfer processing. The packet processor, shown in Figure 7, calls the Token Formation state machine, shown in Figure 8.

---

**Figure 7** • Packet Processing State Graph

---

5-30
The query timer activates the Connection Polling state machine, shown in Figure 9. The polling broadcasts a default address for response by any device new to the network. A response is relayed to the host so that the device can be configured and its entry made in the connection table.

Transmission control is divided into two parts; Packet processing has overall responsibility, and the Token Formation state machine controls the flow of bytes making up the token.

The Token Formation state machine increments the token buffer address counter to point to the next byte so that it is available for transmission before the last bit of the previous byte is shifted out (indicated by the DONE condition in the state graph). On the next clock, the first bit of the next byte begins transmission.

The USB host must support different data clock rates from its devices. The system contains a clock generator counter whose outputs decode to the USB data rates. Data describing clock rate comes from the connection table and is registered to select the appropriate clock frequency for the session.

**FPGA Design**

All state machines are implemented in the FPGA by using bit-per-state encoding. This encoding scheme uses one flip-flop for each state, minimizing combinatorial logic resource requirements. Bit-per-state encoding takes advantage of the balance of sequential and combinatorial resources on the device. It is also the highest-performing encoding scheme.

Data is clocked in and out of the device at a rate determined by the device definition. The frequencies of the possible clocks have not been defined as of this writing, but the variability will require the clocks to be generated within the FPGA, making the USB interface a multiple-clock design.

The 3200DX family features quadrant clocks, which are suited for the multiple and dynamic frequencies requirements described here. Logic common to some clock frequency may reside in one or more quadrants and be clocked by those quadrants’ clocks. Other logic using different clocks may reside in another quadrant.

The existence of dedicated, dual-port, synchronous SRAM resources on the device allows easy implementation of FIFOs for bidirectional communication with the USB and the host system main memory. The FIFO buffer storage uses the dedicated, dual-port SRAM for message storage. Because reads and writes are completely separate, the FIFO controller may allow data from one side of the buffer to be transmitting while the other is receiving.

The FIFO design need not be specified, since it is generated automatically by the Actel ACTgen Macro Builder. The tool will create the FIFO once you enter its size and features. The netlist will be integrated with the top-level design during compilation. The netlist may also be converted to the design in HDL and referenced as a component for simulation.
The connection table and other register functions may also be implemented in synchronous SRAM where addresses are registered and outputs change on clock edges. That allows built-in latency on accesses so that the data in a token field can be shifted out from the Memory Output register while the address for the next is used to access the memory. By this mechanism, the last bit from 1 byte is succeeded by the first bit from the next byte on the next clock edge.

The entire USB interface fits easily into a midsize 3200DX device, leaving resources for the DMA and other host interface functions.

**Conclusion**

New interface standards are popular targets for implementation in FPGAs. The Universal Serial Bus (USB) is a popular emerging interface standard and is easily implemented in Actel FPGAs. In particular, the 3200DX family offers the speed, capacity, and on-chip FIFO capability required by USB. This application note along with the complete design files and test cases (in both schematic and HDL) are available from Actel. Visit the Actel World Wide Web site (www.actel.com), or contact Actel Technical Support.