Using High-Speed Serial Links to Supplement Parallel Data Buses

Today's designers face a multitude of problems when trying to move data within their systems. These problems range from overtaxed parallel-bus bandwidth to a lack of pins at the card edge connector. Even routing parallel buses around today's dense circuit boards is very difficult. This application note discusses using high-performance serial links as a solution to some of these bottlenecks. A serial approach provides three immediate benefits: first, bandwidth may be off-loaded from the backplane bus; second, connector pins are saved; and, third, circuit board routing is made much easier since only one or two traces have to be routed for the data path (versus one for each data bus bit plus a separate clock).

The ideal serial-interface building-block chip set would consist of matched high-speed parallel-to-serial and serial-to-parallel converters (also referred to as transmitter and receiver). Additionally, this chip set would make the serial interface transparent to the user, i.e., parallel data would flow in one side and out the other. It would be able to use a variety of serial media directly such as coaxial or twisted-pair cables, or fiber-optic media (when coupled through the proper optical driver/receiver). It would also be easily adaptable to user-defined protocols for applications involving Direct Memory Access (DMA).

HOTLink™

Cypress's serial interface building blocks for these applications are the CY7B923 and CY7B933 HOTLink Transmitter and Receiver. These devices provide data rates of 150 to 400 Megabits/second (15 to 40 Megabytes/second) and conform to several communications standards.

This application note focuses on utilizing HOTLink to move data using a simple protocol. A block diagram of a typical HOTLink interface is shown in *Figure 1*.

Preliminaries

In this application the basic assumption is that the serial links in question do not exceed approximately 1 meter. This length is adequate for most intra- and inter-board communications situations. Limiting the design to these distances removes several communications system issues from those that must be considered.

HOTLink

In Encoded Mode, HOTLink operates with an 8-bit parallel data path. Data bytes are encoded into 10-bit transmission characters using 8B/10B encoding. In Bypass mode, HOTLink uses a 10-bit data path. 10-bit characters bypass the encoder and go directly to the serializer.

The 8B/10B code is used for most applications because it guarantees enough signal transitions on the serial interface to ensure proper PLL operation. It is also DC balanced, which prevents development of DC-offset in the link over time. DC offset results from more 1s being transmitted than 0s, so the encoder maps the 8-bit input word to multiple 10-bit output values to keep the number of 1s and 0s constant over time. Ideally, the time-averaged DC component should be zero, since DC offset over a long cable can cause increased noise susceptibility.

The application described here uses 8-bit Encoded Mode. In this mode HOTLink provides two control pins: SC/$\overline{D}$ and SVS. SC/$\overline{D}$ is used to indicate if the byte on the parallel I/O pins is a special character or data. SVS (Send Violation Symbol), allows the data provider to force a violation symbol to be encoded and sent. The SC/$\overline{D}$ pins is used to signify command characters in the link protocol, which are described later. The SVS (RVS in the receiver) pin can be used for system testing and error checking, but is not used in this design.

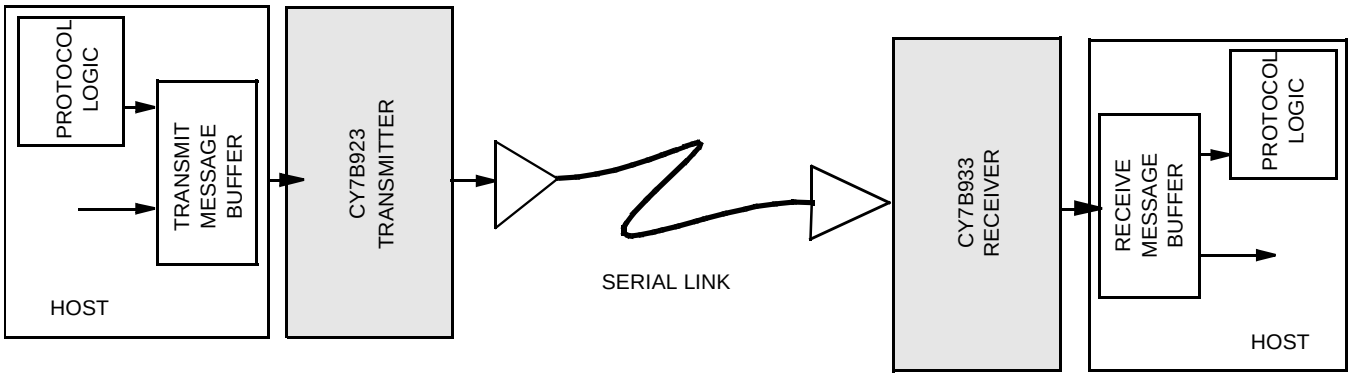

Figure 1. HOTLink System Connections

Parallel Interface

For details of the HOTLink parallel interface, please refer to the CY7B923/CY7B933 HOTLink datasheet.

Transmitter

The signals needed for the transmitter parallel interface are the 8-bit parallel inputs D(0..7), SC/$\overline{D}$, $\overline{ENA}$, $\overline{RP}$, and the CKW clock.

When no data is enabled into the transmitter, the HOTLink Transmitter inserts a special character called SYNC or K28.5. This SYNC character provides sufficient transitions to keep the receive PLL locked to the bit stream.

Receiver

The signals needed for the receiver parallel interface are the eight parallel data outputs D(0..7) and the SC/$\overline{D}$, $\overline{RDY}$, and CKR recovered clock.

The receiver decodes each character as it is received and presents it on the output data bus.

Serial Interfaces

Figures 2 and *3* show the triple serial outputs of the transmitter and the dual serial inputs of the receiver. OUTC is always on and, in full duplex implementations, can be "looped back" to the local receiver for system diagnostics or used as another output. The other output pairs, OUTA and OUTB, are enabled with the FOTO pin. Each output pair may be used to transmit to multiple destinations, making point-to-multi-point DMA architectures possible.

The receiver, with its pair of inputs, can use one input channel for data and the second to implement local loopback. The input selection is controlled by the A/$\overline{B}$ pin. Note that the INB channel does not have to be used for diagnostics, but can be used as another data stream input. However, switching input channels requires the PLL to reacquire lock and framing with the incoming data stream.

Implementing a Data Link

The following discussion deals with issues confronting a designer trying to move data from point A to point B using HOTLink. *Table 1* shows the three implementations discussed.

I/O Space Model

The first example assumes that a receive FIFO resides in the destination's I/O space. The receive controller (a microprocessor, or state machine) merely reads and interprets the data stream out of the RX FIFO. Data does not get placed in local memory before being used. The main concern here is to make sure that the receive logic can keep up with the received data stream. Failure to do so will cause a receiver overflow or transmitter overrun. A FIFO with programmable almost full/empty flags can be used together with a PLD to generate an inhibit to the transmit controller. This is known as "flow control."

Figure 4 shows a receiver block diagram with a flow control signal labeled TXINH. If the FIFO becomes too full, this signal tells the transmitter to stop transmitting until the receiver catches up. Since we are limiting our links to three or four feet, this may be a viable approach. However, using this form of flow control wastes a lot of bandwidth. Correct sizing of the FIFO, after careful analysis of the communications requirements, can yield a deterministic system that never overflows or overruns. Communications links of hundreds of feet, or even miles cannot afford this type of flow control, since the channel itself may be several hundreds or even thousands of bytes long and a large enough FIFO may not be available. The channel is like a pipeline, and once something enters the pipeline, it must come out the other end.

The second issue is that the microprocessor must be interrupted when data needs to be read from the FIFO. This wastes processor cycles and adds to system latency.

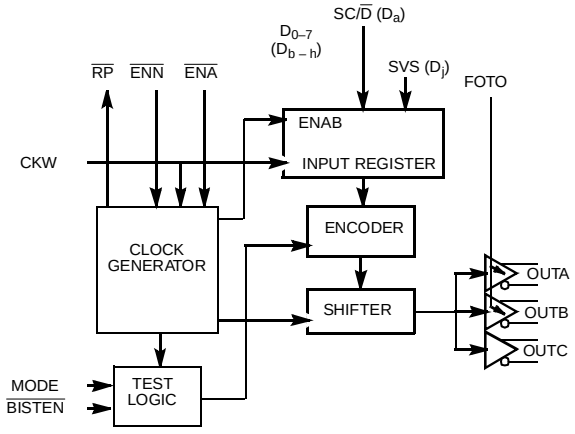

Figure 2. CY7B923 Transmitter Logic Block Diagram

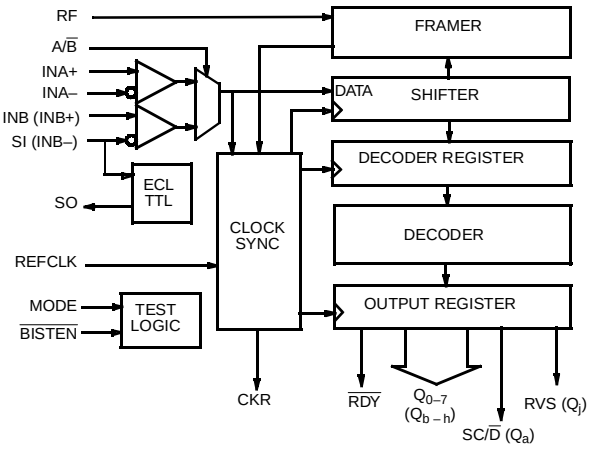

Figure 3. CY7B933 Receiver Logic Block Diagram

Table 1. Data Link Implementations

I/O Model	Transmitter	Receiver	Features
I/O Space	FIFO+HOTLink	HOTLink+FIFO+ Microprocessor	RX FIFO in I/O Space Microprocessor Accesses Data
Direct Memory Access	FIFO+HOTLink	HOTLink+FIFO+ DMA Logic	RX Controller Decodes DMA Info in RX FIFO DMA Moves Data Microprocessor Free Local Bus Used
Shared Memory Space	FIFO+HOTLink	HOTLink+DUALPORT+ DMA Logic	RX Controller Uses Semaphores to Move Data Directly to Shared Memory Microprocessor Free/Local Bus Free

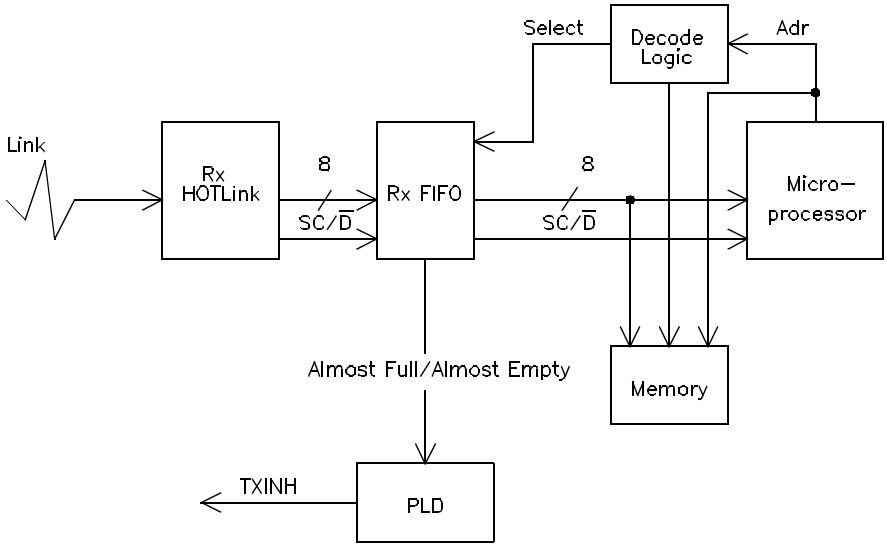

Figure 4. Receiver Flow Control

Direct Memory Access Model

The I/O Space model discussed moving data from Point A to Point B using a microprocessor. Direct Memory Access (DMA) uses additional hardware, called a DMA controller or DMA Logic, to move the data from the FIFO to the memory. This frees the microprocessor from this task. With character cycle times as short as 25 ns (for a 40-MHz character rate), DMA logic that moves the data without the CPU is much more efficient.

The DMA Logic contains several basic functions:

- control state machine
- address counter
- address (and sometimes data) latches and drivers

The control state machine detects when the receive FIFO contains data and issues a DMA request to the microprocessor. The DMA request asks the microprocessor to relinquish the memory address and data buses. The DMA state machine also detects when the microprocessor has freed the buses. It then starts the actual transfer by loading the address counter with the initial memory address and enables the initial address and data onto the memory address and data buses. The control state machine then strobes the data into the memory, increments the address counter, reads the next data from the receive FIFO onto the memory data bus and strobes this data into the latches. This process repeats until all the data has been transferred to memory. Upon completion of the transfer, the memory address and data buses are returned to the microprocessor's control. A block diagram of a DMA system is shown in *Figure 5*.

In this type of link the system must manage the start and destination locations in the system memory. To do this correctly requires that a protocol be defined.

DMA Protocol Definition

The DMA protocol requires embedding command and control information in the data stream. Previously the information consisted of pure data (as far as the control logic was concerned). The data movement is now managed by dedicated hardware (a PLD or other DMA controller) to move the data from the RX buffer to the memory location where it will be used, thus off-loading the main processor.

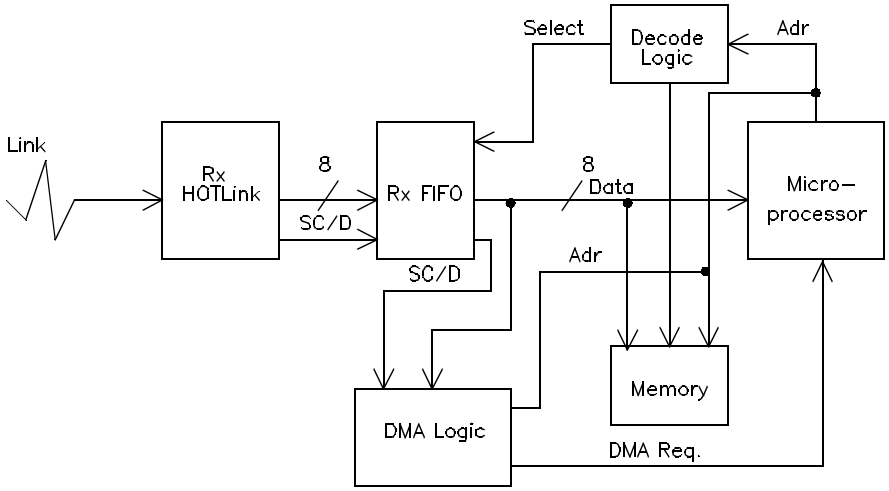

Figure 5. DMA Configuration

For the examples used in this application note, the same protocol definition is used for both the DMA and the shared memory implementation. Both of these assume a fixed length DMA transfer of 256 bytes. (The user is free to implement any length required, or to provide for variable length transfers.)

Table 2 defines a DMA Write message. The protocol consists of a special character or message delimiter signifying a DMA write request, followed by characters defined as a broadcast address, and a starting address. A broadcast address can be thought of as a card or processor ID. The starting address indicates the first memory address to be written. This is followed by N-bytes of data, where N is equal to 256 in this example.

DMA reads are identified by a unique message delimiter as shown in *Table 3*. Again, this example assumes that 256 bytes of data are requested. The message defined in *Table 3* tells the recipient to send the 256 bytes of data beginning at the indicated address. It also provides a destination address that can be used to create the DMA write message.

Finally, to assure proper initialization of the DMA hardware, *Table 4* defines a DMA Reset message.

The column labeled “Bits” in *Tables 2*, *3*, and *4* deserves further explanation. The labels HGF EDCBA are the 8B/10B designations for bits on the HOTLink parallel interface. Conventional notation for these bits is Q7..Q0 on the receiver outputs and D7..D0 on the transmitter inputs, with bit 7 being the most significant bit. In fact these signals correspond to the pins labeled identically on the HOTLink devices. The message delimiter characters are named according to Fibre Channel convention. Refer to the CY7B92X/CY7B93X data sheet for additional information.

The receiver DMA Logic in *Figure 5* needs to contain a state machine to detect the appropriate message delimiters and decode the broadcast address. If the broadcast address is for the module and the message is a DMA write, another state machine needs to issue a DMA request to the microprocessor and obtain the bus. After obtaining the bus, the starting address is read from the FIFO and loaded into an address counter. Since the address is defined as 32 bits, and the message length is defined as 256 bytes, the address must be

Table 2. DMA Write Message

SC/$\overline{D}$ Pin	Character Name	Bits (HGF EDCBA)	Definition
1	K28.1	000 00001	Msg. Delimiter
0	8-bit address	–	Broadcast Address
0	Address byte 0	–	Most significant
0	Address byte 1	–	Next most significant
0	Address byte 2	–	Next least significant
0	Address byte 3	–	Least significant
0	Data byte 0	–	1st data byte
0	Data byte 1	–	2nd data byte
0	:	:	:
0	Data byte N	–	Last data byte
1	K28.1	000 00001	Msg. Delimiter

Table 3. DMA Read Message

SC/$\overline{D}$ Pin	Character Name	Bits (HGF EDCBA)	Definition
1	K28.0	000 00000	Msg. Delimiter
0	8-bit address	–	Broadcast Address
0	Source Address byte 0	–	Most significant
0	Source Address byte 1	–	Next most significant
0	Source Address byte 2	–	Next least significant
0	Source Address byte 3	–	Least significant
0	Destination Address byte 0	–	Most significant
0	Destination Address byte 1	–	Next most significant
0	Destination Address byte 2	–	Next least significant
0	Destination Address byte 3	–	Least significant
1	K28.0	000 00000	Msg. Delimiter

Table 4. DMA Reset Message

SC/$\overline{D}$ Pin	Character Name	Bits (HGF EDCBA)	Definition
1	K28.3	000 00011	Msg. Delimiter
0	8-bit address	–	Broadcast Address
1	K28.3	000 00011	Msg. Delimiter

loaded into three latches and an 8-bit counter. Then the state machine reads the next 256 bytes from the receive FIFO and writes it to memory. The bus is then relinquished to the microprocessor, and the counters and state machine are reset.

When the message is a DMA read, the state machine operates similar to that for a DMA write, but the address counter is loaded with the address to read from, and the destination address is read out and placed into a DMA write message. Creation of DMA write messages can be accomplished with additional resources in the DMA Logic. A suggested device is the Cypress CY7C375 CPLD or the Cypress CY7C385 FPGA.

The DMA model off-loads the data moving task from the microprocessor. However, DMA has one major disadvantage; it requires the bus, which means the microprocessor must be idle during the DMA. There is another approach and it is the next topic.

Shared Memory I/O Model

If a shared memory area (dual-ported) is implemented, then data can be made available to the local logic without removing the microprocessor from the memory bus to perform a DMA. Simultaneous access to the same memory location must be prevented, but dual-ported memory opens up several options. These include dividing the dual-ported memory into segments and alternating the segments between DMA write and local side read. This is known as “ping-ponging” and prevents simultaneous access of a dual-port SRAM location.

Dual-ported memory is attractive for those cases where the local bus cannot be tied up with a true DMA. *Figure 6* shows a diagram of a HOTLink communications link implemented with dual-ported SRAM. The DMA Logic is virtually identical to that of the prior section.

Summary

This application note presents the basic concepts for employing HOTLink high-speed serial communications devices to replace parallel data paths. It also defines a simple protocol and describes the logic necessary to implement that protocol. Finally, the advantages and disadvantages of three different approaches have been presented to allow the designer to choose the one that best fits their needs. The simplest is Memory Mapped I/O, the highest performing is the Shared Memory I/O model employing dual-ported memory.

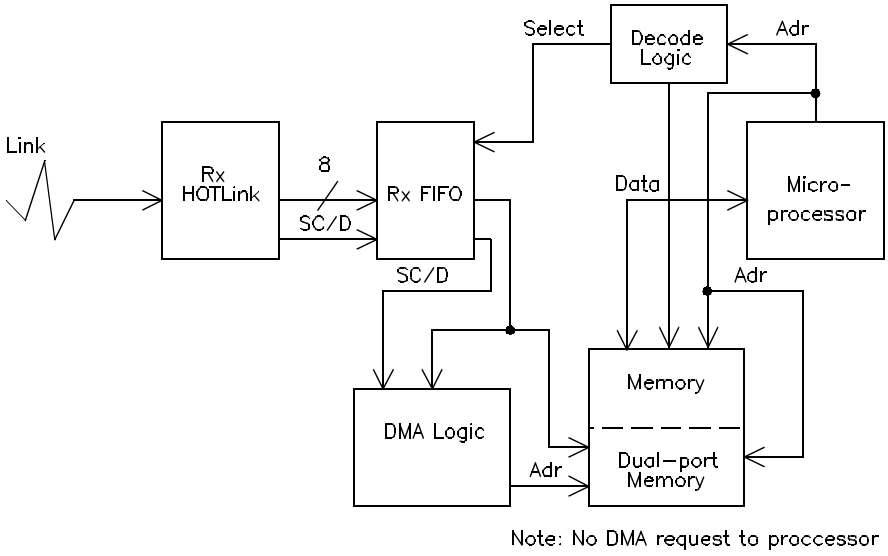

Figure 6. Dual-Ported Configuration