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 offloaded from the backplane bus; second, connector pins are saved; and, third, circuit board routing is made much easier since only two traces have to be routed for the data path (versus one for each data bus bit).

The ideal serial interface building block would be a chip set consisting of 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 cable, twisted-pair cable or even fiberoptic cable (when connected to the proper optical driver). It would also be easily adaptable to user-defined protocols for applications involving Direct Memory Access (DMA).

HOTLink™

Cypress's serial interface building blocks are the CY7B923 HOTLink Transmitter and CY7B933 HOTLink Receiver. These devices provide data rates of 160 to 330 Megabits/sec (16 to 33 Megabytes/sec) 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*.

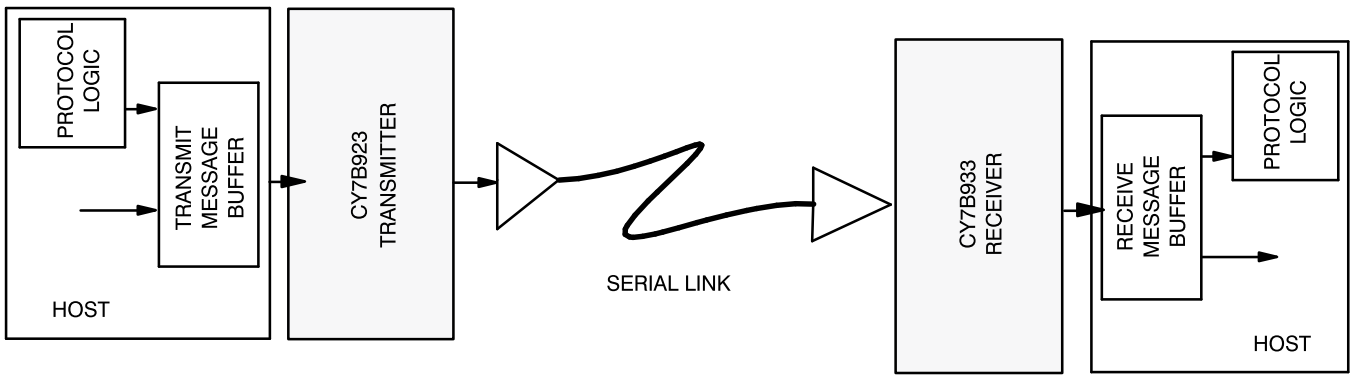

Figure 1. HOTLink System Connections

Preliminaries

For this application we will assume that the serial links in question will not exceed three or four feet. This length is adequate for most intra-board and board-to-board communications situations; and limiting ourselves to these distances removes several communications system issues from those that must be considered. Let's now discuss the general features of the HOTLinks.

In Encoded Mode, the HOTLink has an 8-bit parallel interface. Data bytes are encoded into 10-bit transmission words using 8B/10B encoding. In Bypass mode, HOTLink uses a 10-bit interface. 10-bit words bypass the encoder and go directly to the serializer. The 8B/10B code provides 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 can result 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 1's and 0's constant over time. Ideally, the time-averaged DC component should be zero, since DC offset over a long cable can cause increased noise susceptibility and power dissipation. In fiber systems excessive DC offset can burn out the LEDs used to drive the fiber.

The applications in this note use 8-bit Encoded Mode. In this mode HOTLink provides two control pins. A pin called SC/$\overline{D}$ indicates whether the byte on the parallel I/O pins is a special character or data. Another pin available in 8-bit mode, SVS (Send Violation Symbol), allows the data provider to force a violation symbol to be encoded and sent. The SC/$\overline{D}$ pins will be used to signify command words in the DMA protocol, which will be specified later. The SVS (RVS in the receiver) pin could be used for system testing and error checking, but will not be part of the design.

Parallel Interface

For details of the HOTLink parallel interface, please refer to the FIFO-HOTLink application notes located in this book.

Transmitter

The signals needed for the transmitter parallel interface are the 8-bit parallel inputs D(0..7), the SC/$\overline{D}$ bit, the $\overline{ENA}$ pin, the $\overline{RP}$ pin, and the CKW pin. Refer to the CY7B923/CY7B933 HOTLink datasheet for additional details.

When no data is enabled into the transmitter, it should be noted that the HOTLink Transmitter inserts a special character called SYNC. This SYNC character provides sufficient transitions to keep the PLLs 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 pins. Again, refer to the device datasheet for additional details.

When the transmitter is sending SYNC characters, the receiver detects these and does not output this character until the last SYNC character is received. Then the receiver outputs a single SYNC character.

Serial Interfaces

Figure 2 shows the multiple 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 a receiver input for system diagnostics or used as another output. The other pair of outputs, OUTA and OUTB are enabled with the FOTO pin. This output pair makes it easy to transmit from one source 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 accomplished with 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 with the incoming data stream.

Implementing a Data Link

The discussion that follows deals with issues confronting a designer trying to move data from point

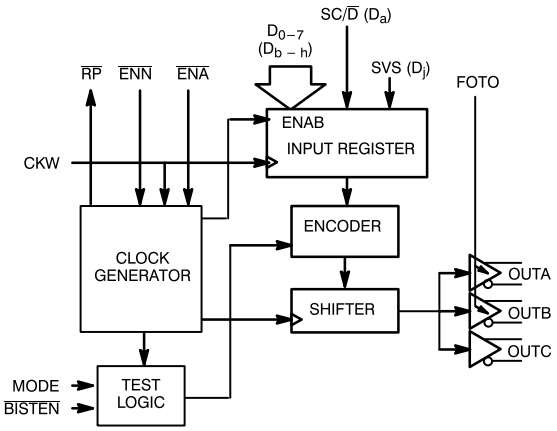

Figure 2a. CY7B923 Transmitter Logic Block Diagram

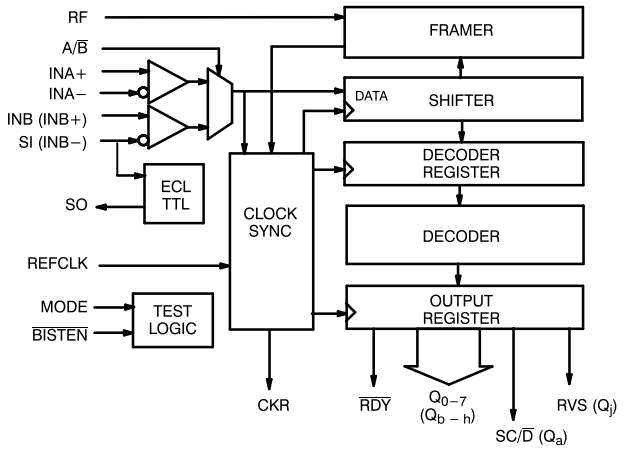

Figure 2b. CY7B933 Receiver Logic Block Diagram

A to point B using HOTLink. *Table 1* shows the three implementations discussed.

I/O Space Model

The first example is simple. It assumes that the receive FIFO resides in the destination's I/O space. The receive controller (a microprocessor, perhaps) merely reads and interprets the data stream out of the Rx FIFO. Data does not get placed in local memory before being used. There are two issues to consider with this example: What if the receive logic cannot keep up with the received data? This is known as receiver overflow or transmitter overrun. A FIFO with a programmable almost full/empty flag can be used together with a PLD to generate a receive or transmit inhibit to the transmit controller. This is known as "flow control". *Figure 3* 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 give a deterministic system that never overflows or underflows. 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.

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

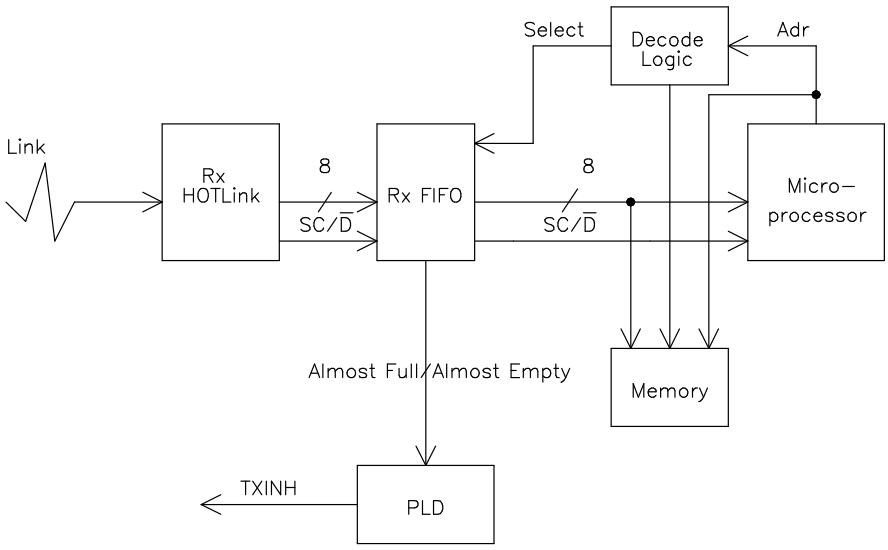

Figure 3. Receiver Flow Control

The second issue is that the microprocessor must be interrupted when data needs to be read from FIFO. This wastes microprocessor cycles and required lots of latency.

Direct Memory Access Model

So far we have 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 of this task. Before proceeding, let's look at the bandwidth supported by HOTLink. This will determine the speed at which our DMA logic must operate. Refer to *Table 2*.

Table 2. Bandwidth Requirements

Part Number	Bandwidth (MByte/s)	Clock Period (ns)
CY7B923/933	16 – 33	63 – 30

Table 2 shows that DMA is probably a better solution than the I/O space model. The DMA Logic contains several basic functions. These are:

- 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 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 address and placing the initial address and data word on 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 memory. The process then repeats until all the data has been placed in memory. When all the data has been placed in memory, the memory address and data buses are returned to the microprocessor's control. A block diagram is shown in *Figure 4*. Questions at this point might include, "Where did the starting address come from?", or "Where did the ending address come from?" To answer these questions, a protocol needs to be defined.

DMA Protocol Definition

Let's now discuss some concepts being introduced with our DMA protocol definition. First, we are now embedding command and control information in the data stream. Previously the information consisted of pure data (as far as our control logic was

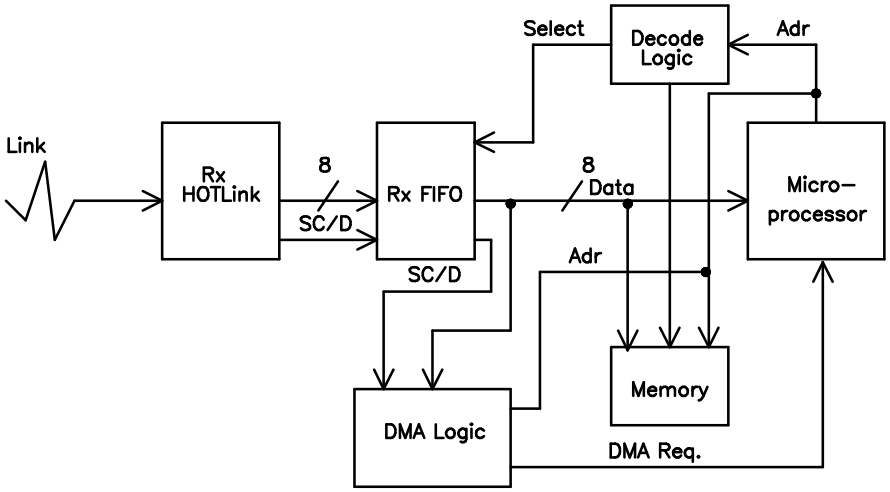

Figure 4. DMA Configuration

concerned). Second, we are now using dedicated hardware (a PLD) to move the data from the rx buffer to the location where it will be used, thus off−loading the main processor. Since the same protocol definition will be used for the shared memory implementation, let's define a protocol. First, the design will assume a fixed length DMA of 256 words. (The user is free to implement any length required, or to provide for variable length transfers.)

Table 3 defines a DMA Write message. The protocol will consist 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 address to be written. This is followed by N−bytes of data, where N is equal to 256 in the example.

DMA reads will be identified by a unique message delimiter as shown in *Table 4*. Again, it is assumed that 256 bytes of data will be sent. The message defined in *Table 4* tells the recipient to send the 256 bytes of data beginning at the address indicated, and 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 5* defines a DMA Reset message. The column labeled "Bits" in *Tables 3*, *4*, and *5* 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 datasheet for additional information.

The receiver DMA Logic in *Figure 4* 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 loaded into 3 latches and an 8-bit counter. Then the state machine to reads the next 256 bytes out of the receive FIFO and writes it to memory. Then the bus is relinquished to the microprocessor and the counters and state machine are reset.

When a the message is a DMA read, the state machine is similar to that for a DMA write, but the ad-

dress counter is loaded with the address to read from, and the destination address is read out to be placed into a DMA write message. Creation of DMA write messages can be accomplished with additional resources in the DMA Logic. Suggested devices are the Cypress CY7C375 CPLD or the Cypress CY7C385 FPGA.

Table 3. DMA Write Message

SC/$\overline{D}$ Pin	Byte 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 signif.
0	Address byte 2	–	Next least signif.
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 4. DMA Read Message

SC/$\overline{D}$ Pin	Byte 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 signif.
0	Source Address byte 2	–	Next least signif.
0	Source Address byte 3	–	Least significant
0	Dest. Address byte 0	–	Most significant
0	Dest. Address byte 1	–	Next most signif.
0	Dest. Address byte 2	–	Next least significant
0	Dest. Address byte 3	–	Least significant
1	K28.0	000 00000	Msg. Delimiter

Table 5. DMA Reset Message

SC/$\overline{D}$ Pin	Code 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

The DMA model offloads 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 grabbing the microprocessor bus to perform a DMA. Simultaneous accesses must be prevented, but dual-ported memory opens up several options. These options 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. So, dual-ported memory is attractive for those cases where the local bus cannot be tied up with a true DMA. *Figure 5* 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 has presented the basic concepts for employing HOTLink high-speed serial communications devices to replace parallel data paths. It has also defined a simple protocol and described the logic necessary to implement the 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.

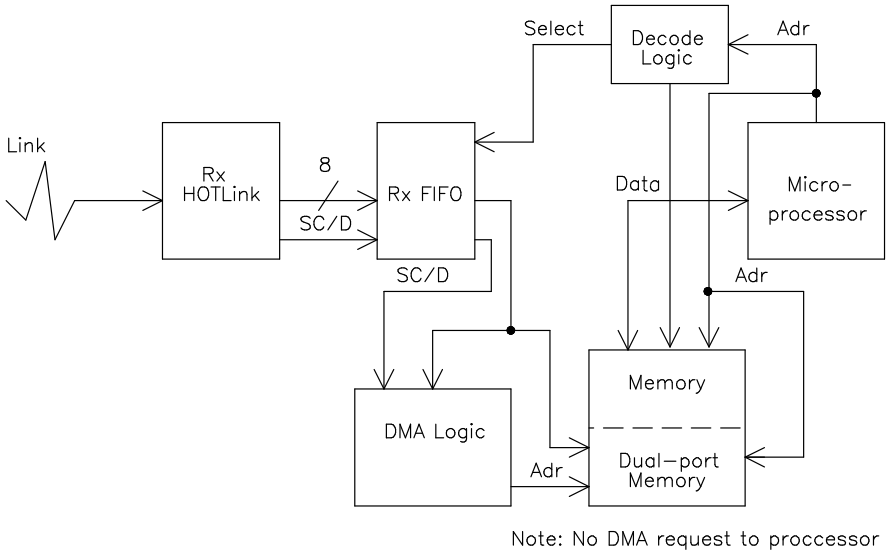

Figure 5. Dual-Ported Configuration

HOTLink is a trademark of Cypress Semiconductor Corporation.