Multiplex Serial Interfaces With HOTLink™

Introduction

Serial interfaces have been used for digital communications almost as long as digital logic has been in existence. By far the largest majority of these serial interfaces operate at what are today considered to be relatively slow speeds. This would include computer interfaces like RS-232C/V.24 and RS-422/V.35, and telecommunications interfaces like ISDN and T1/E1 digital telephone lines. Whenever more than one of these types of interfaces must be connected between two points, it usually requires a separate electrical or optical cable for each serial link.

Many installations require large numbers of serial interfaces to be connected between various peripherals and computer systems. At times these interfaces must be routed across distances or environments that are not normally supported by the native interface. To save on cable and installation cost, and the physical space required for multiple cables, multiplexers were developed that allow more than one serial interface to communicate across a common cable. These multiplexers exist in a wide range of functionality and complexity.

This application note describes the architecture of standard types of serial multiplexers, describes a new architecture for a high-performance, low-cost multiplexer, and shows a complete design based on this architecture.

Statistical Multiplexers

A popular type of multiplexer for use with RS-232C serial communications is known as a statistical multiplexer. It operates by accepting bytes of data from multiple slow-speed serial sources, and combining them into a single, faster serial-rate interface.

The word *statistical* is important here. In this type of multiplexer, the high-speed serial interface does not have sufficient bandwidth to allow all input ports to operate at full-speed with sustained data transfers. It instead operates on the statistical probability that few of the slow-speed ports will need to transfer information at the same instant in time. When the available bandwidth of the high-speed link is consumed, the remaining slow-speed ports are forced to wait until some bandwidth becomes available.

This type of multiplexer is generally sufficient for text-based keyboard entry and displays, but not for telemetry or machine control applications. These applications require an interface that can provide a deterministic response with a guaranteed bandwidth.

Time Division Multiplexers

Time division multiplexing operates by assigning a fixed portion of the total bandwidth of the high-speed link to each of the slower-speed serial interfaces. The bandwidth allocation is usually made to allow each of the slower-speed interfaces to operate at their maximum rate. The multiplexing is often performed at a byte level, with various link-level protocols running to allow the merged data streams to be correctly separated at the destination end of the link.

Multiplexers of this type are generally limited by the maximum bandwidth available on the high-speed serial link. Because of this limited bandwidth, multiple forms of re-synchronization and compression are often added to maximize the usage of the available bandwidth. Unfortunately these also add significant cost and complexity to the total system.

Serial Interface Multiplexer

With the development of low-cost, high data-rate serial interface ICs, a simpler method is now available to create a multiplexed serial interface. This method requires no interpretation of any of the data present at the input or output serial interfaces, and is both self synchronizing and initializing.

This type of multiplexer operates by sampling serial or static input signals at a very high rate, with these samples then routed across a high-speed serial link to a remote register where they are then output as a continuous sequence of bits. A multiplexer of this type, supporting eight bidirectional serial streams, can be made using a pair of CY7C371 CPLDs and two CY7B923/CY7B933 HOTLink™ transmitter/receiver pairs. *Figure 1* shows a block diagram of a multiplexer based on this architecture.

Because of the sampled nature of this type of design, each serial input can operate at a different rate. These streams can run as slow as DC, and (if operated synchronous to the byte-rate reference clock) as fast as 40 Mbits/second!

The level translation block in *Figure 1* is used to convert the serial input signal to the TTL domain of the CPLD. Most serial interfaces use logic levels other than standard TTL or CMOS. Common signaling for serial interfaces use current-loop, RS-232C, RS-449, and RS-485 signal levels. To allow these signals to be sampled prior to serialization, they must be converted to the TTL domain.

Serial Sampling Flip-Flops

Following conversion, the signals are sampled at a rate much faster than the native bit-rate of the incoming data. The byte-rate clock used to load data into the HOTLink transmitter is also used as the sampling clock. This allows the incoming data to be sampled at between 16 and 40 Megasamples per second. Since the data is asynchronous to the first sampling register, the output of each sampling register is then routed through a second flip-flop to remove any metastability effects from the sampled data. This serial input-to-HOTLink transmitter data path is shown in *Figure 2*. The logic in *Figure 2* is actually duplicated eight times, once for each serial input stream.

When selecting a sampling clock rate, keep in mind how the data will be used at the remote end of the link. For most asynchronous data streams, the data recovery operation is performed by a UART (Universal Asynchronous Receiver/Transmitter). These UARTs also use oversampling to recover the

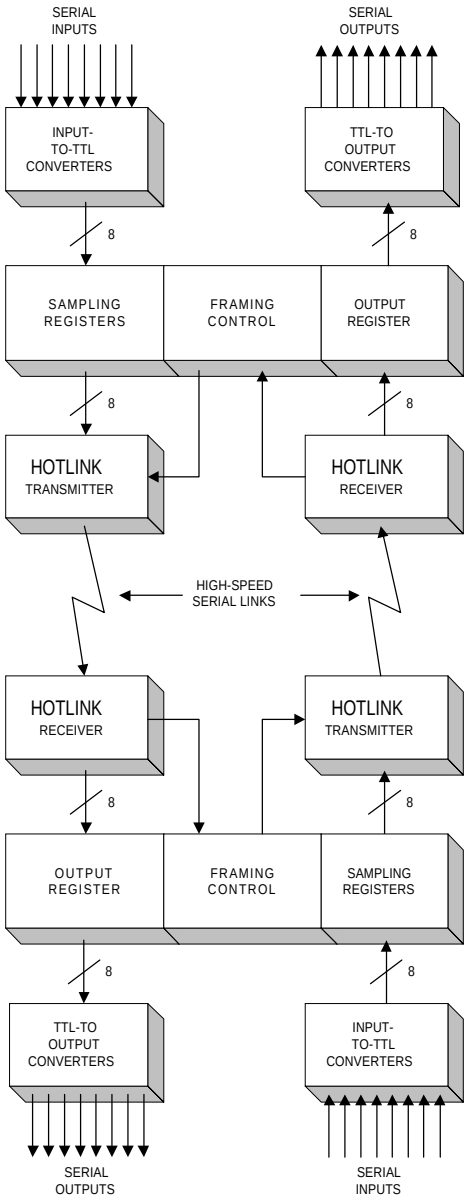


Figure 1. Serial Multiplexer Block Diagram

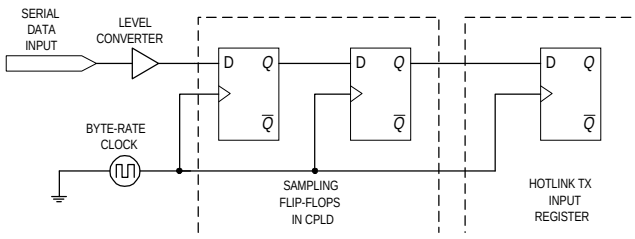


Figure 2. Single-Bit Transmit Path

data. This oversampling is usually done at a rate 16 times the bit-rate of the data being received.

To limit any edge displacement effects caused by the sampling flip-flops in the multiplexer, the serial streams should also be sampled at a minimum of 16 times the fastest bit-rate being sent. Even when sampled at this fast rate, this multiplexer design allows each link to support data rates up to 2.5 Mbits/second.

High-Speed Serial Links

As each sample clock occurs, a group of eight samples (one from each serial stream) is presented to the CY7B923 HOTLink transmitter. The transmitter is configured in ENCODED mode, with $\overline{\text{ENA}}$ normally active (LOW). This allows the HOTLink transmitter to accept a byte of data on each rising clock edge, encode the data using its integrated 8B/10B encoder, and sequence out a stream of bits at ten times the sample clock rate.

These bits are then sent across a fiber-optic, coaxial, or twisted pair cable to a remote CY7B933 HOTLink receiver. The details of how to couple and communicate with HOTLink using various types of media are covered in the Cypress *HOTLink User's Guide*.

The HOTLink receiver is configured with the same byte-rate reference clock as the transmitter. Its internal PLL (Phase Locked Loop) based data-separator allows it to extract a bit-rate clock from the serial data stream, and use this clock to recover the bits sent across the serial interface.

Framing Control

To decode characters from the serial data stream the HOTLink receiver needs to know where the characters begin and end. In asynchronous serial interfaces, the start of a character is usually indicated by the first transition detected by the receiver. This transition marks the first bit of the character and is often known as a *start-bit*.

HOTLink operates with a synchronous serial interface where bits are always being transmitted. Rather than using a start-bit to determine the first bit of each character, a 10-bit counter is used to count off the bits in each character. As the serial bits are received they are clocked into a shift register. When the 10-bit counter rolls over, the bits in the shift register are captured as a 10-bit character. This 10-bit character is then decoded into the original byte that was transmitted using the integrated 8B/10B decoder in the HOTLink receiver.

To properly decode the data, the HOTLink receiver must also be correctly framed to the incoming serial data stream. Framing is the alignment of the internal 10-bit counter with the data on the serial interface. Until the receiver is correctly framed, it has no way of knowing the starting point of a character in the data stream, and many received groups of ten bits will decode into illegal characters. To frame to the data stream it is necessary for framing to be enabled (RF set HIGH) and for one or more K28.5 characters to be received.

The K28.5 (or SYNC) character is a special group of ten bits that cannot occur within or across any other group of ten bits. Since this pattern is unique, the receiver can use it to determine the beginning and ending points of a character. When a SYNC character is received (and framing is enabled), the 10-bit counter in the receiver is reset so that all following characters are counted off from the correct location in the bit stream.

For use in links of this type, the HOTLink receiver contains a multi-byte framer to prevent framing on received characters that have been modified (by noise or other external effects) to look like a SYNC character. When the multi-byte framer is

enabled, the receiver requires a minimum of two SYNC characters within a five-byte span to allow it to frame to the data stream.

Framing Controller

The information sent across the high-speed link is normally all data characters. Once a link has been properly framed, there is no need to send additional synchronization characters unless some external event causes a receiver to lose framing. A well-designed link running in a normal environment should be able to maintain synchronization over a period of months or even years.

However, external events will eventually occur: cables are unplugged, power is lost. When events like these happen (including first power on) the HOTLink receiver no longer knows where each 10-bit character starts or ends. The only way to correct this is for the HOTLink transmitter to send SYNC characters instead of data characters, allowing the receiver to frame to the data stream.

By making use of the automatic SYNC generation and detection capabilities built into the HOTLink transmitter and receiver, it is possible to build a small state machine to monitor the high-speed links, and automatically generate framing characters when needed. A state diagram of such a framing controller is shown in *Figure 3*, with its operation detailed in *Figure 4* and the following text. The VHDL source code for this machine is listed in process PROC1 in *Appendix B*.

> NOTE: The signal, data, and state relationships shown in *Figure 4* are relative in nature. The time relationships between a signal and its resulting action may take more clocks than shown, but will never take fewer.

This state machine operates along two different paths. One path is entered if a character error is detected (indicating that the local receiver may be out of lock), while the other path is entered if a burst of SYNC characters is detected (indicating that the remote receiver needs to be reframed).

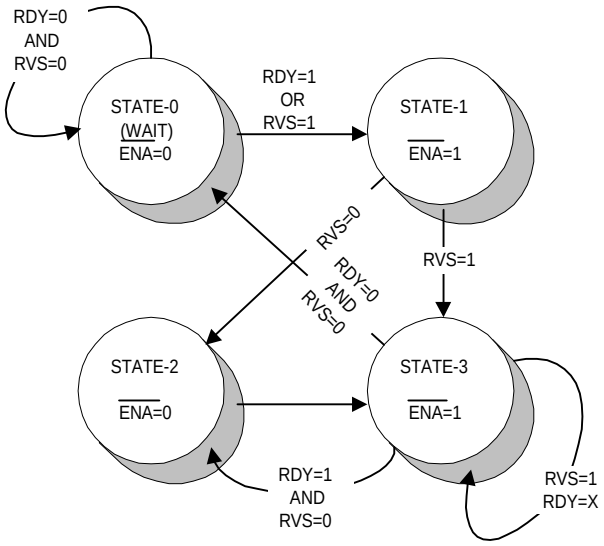


Figure 3. Framing Controller State Diagram

All character errors are indicated by a HIGH state on the HOTLink receiver RVS signal. However, SYNC characters are detected in one of two different modes. If a single SYNC character is detected (one that has a data character on either side of it), that SYNC character is presented to the HOTLink receiver outputs and $\overline{RDY}$ pulses LOW (the same as with any other character). However, if multiple contiguous SYNC characters are received (with no data or non-SYNC characters

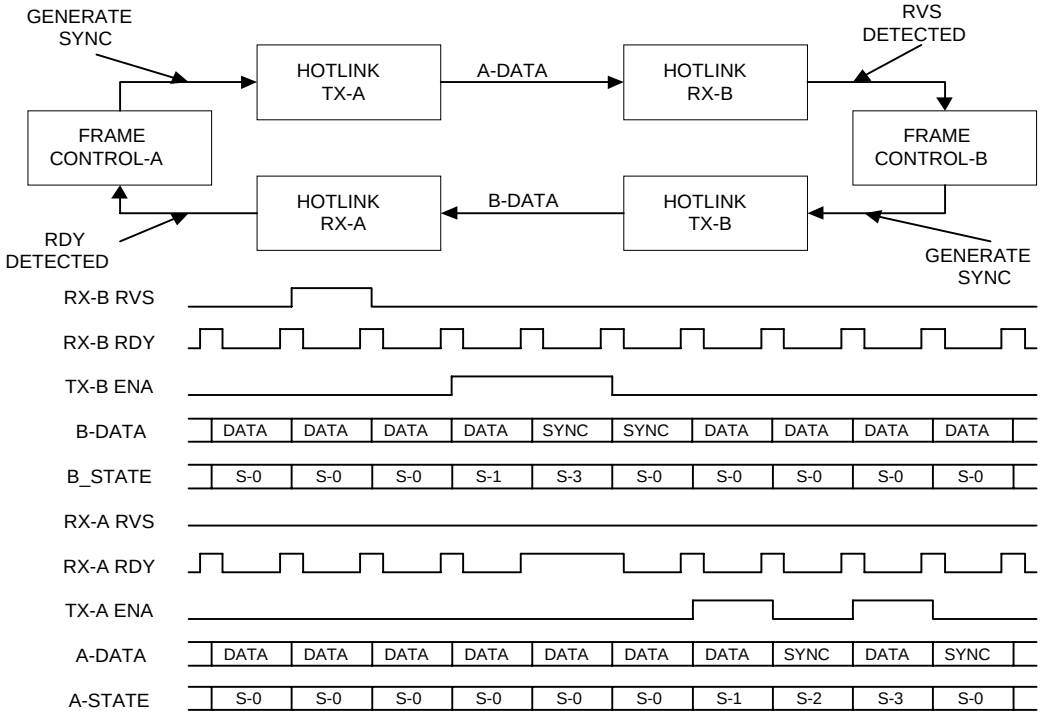


Figure 4. Framing Controller Operation

between them) then the $\overline{RDY}$ signal will go HIGH and stay HIGH for all but the last SYNC character received.

Operation

The frame-control state machine at both ends of each link normally waits in State-0, looking for either the RVS or $\overline{RDY}$ pins on the attached HOTLink receiver to be HIGH at the rising edge of the recovered clock (CKR). If RVS is HIGH, the receiver has detected a character that violated one (or more) of the encoding rules for 8B/10B characters. This generally means that either the character was corrupted during transmission, or that the receiver is no longer correctly framed to the data stream. When detected by the receiver at the B end of the A-to-B link shown in *Figure 4*, the status of RVS being active is latched into Frame Controller-B, and the Frame Controller-B state machine moves to State-1.

If $\overline{RDY}$ were HIGH when in State-0, the state machine would also advance to State-1. $\overline{RDY}$ being HIGH indicates that two (or more) directly adjacent SYNC characters were detected by the local HOTLink receiver. In *Figure 4*, the receiver at the A end of the B-to-A link is shown driving $\overline{RDY}$ HIGH for one byte time. This causes the Frame Controller-A state machine to also move to State-1.

In State-1, the $\overline{ENA}$ input to the local HOTLink transmitter is driven HIGH. This forces the local transmitter to generate a SYNC character that can then be used by the remote receiver for framing. If State-1 was entered because RVS was HIGH, the state machine advances directly to State-3. If RVS was LOW, then the machine enters State-2. In *Figure 4*, Frame Controller-B moves directly to State-3, while Frame Controller-A moves to State-2.

State-2 is a filler state. No actual decisions are made in this state, but it does control the generation of SYNC characters, and enters State-3 on the next clock. Since $\overline{ENA}$ is set LOW in State-2, the local HOTLink transmitter encodes and sends whatever character is presented to its input register, instead of sending another SYNC character. This data character, sent immediately following a SYNC character, allows the remote receiver to start a reframe operation, but will not generate a $\overline{RDY}$ HIGH indication at the remote receiver. This keeps the remote framing control state machine in the State-0 or wait state.

State-3 is the normal ending state for both the $\overline{RDY}$ and RVS induced paths through the framing control state machine. In State-3, $\overline{ENA}$ is again driven HIGH, forcing generation of a second SYNC character. If the state machine was initially triggered by $\overline{RDY}$, a pair of SYNC codes are generated, but with a single byte of data sent between them. When these SYNC codes are detected by the remote receiver, it will (if necessary) adjust its framing to the correct character boundaries. Since a data character was sent between the two SYNC characters, the remote receiver will not generate a $\overline{RDY}$ HIGH condition.

If the machine was triggered by RVS, a pair of directly adjacent SYNC codes are generated. When these codes are detected by the remote receiver, it will (if necessary) adjust its framing to the correct character boundaries. Since the SYNC codes are directly adjacent, they will also force a $\overline{RDY}$ HIGH indication at the remote HOTLink receiver, triggering the remote framing control state machine.

If RVS remains active in State-3, the machine remains in the same state and continues to generate a continuous stream of SYNC codes. When RVS is finally removed, the machine returns to State-0.

Analysis of the functionality of this state machine shows that it has no dead-lock conditions. This allows bidirectional links to be built with this machine in a peer-to-peer fashion, rather than requiring one end to be declared a master and the other a slave.

Clocks

The output of the reframe control state machine controls the HOTLink transmitter, and therefore must be synchronous to the transmitter CKW clock. Unfortunately, the two status inputs that control the state machine's operation ($\overline{RDY}$ and RVS) are synchronous to the HOTLink receiver CKR clock. To insure that these two signals (which may only be valid at a single rising edge of CKR) are not missed by the reframe control state machine, they are first captured using the CKR clock. Once captured, they are routed through metastable prevention flip-flops to convert them to the CKW clock domain. This is the same CKW clock used for the sampling registers, the HOTLink transmitter byte clock, and the REFCLK for the HOTLink receiver.

The CKR clock, generated by the HOTLink receiver, is based on the clock extracted from the received high-speed serial data stream. This clock, while close to CKW in frequency, is totally asynchronous to CKW. This CKR clock is used to capture the byte-wide recovered data that the HOTLink receiver outputs on every rising edge of the CKR clock.

Serial Output Register

The data decoded by the HOTLink receiver should not be directly fed to the output level converters. The HOTLink receiver output register will at times contain various command characters (in addition to the normal data characters) when errors are detected or when SYNC codes are being received.

To prevent these non-data characters from propagating to the low-speed serial interface outputs, the data is fed from the HOTLink receiver into an output register. This register is configured such that it keeps the its last data value if a command character is present in the HOTLink receiver output latch. Since illegal characters are also decoded as command characters, it is only necessary to use the SC/$\overline{D}$ pin from the HOTLink receiver as a controlling signal. The HOTLink receiver-to-serial output data path is shown in *Figure 5*. The logic in *Figure 5* is actually duplicated eight times, once for each serial output stream.

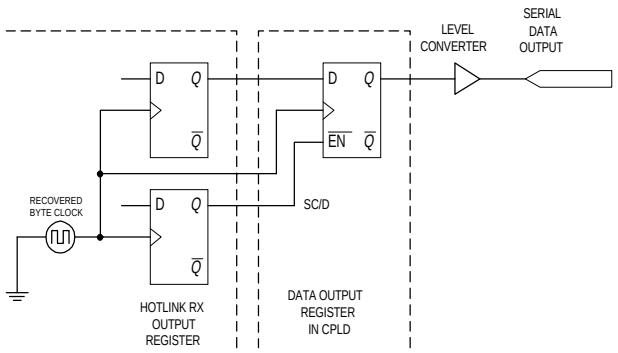


Figure 5. Single-Bit Receive Path

Serial Interface Multiplexer Design

The schematic of one end of such a multiplexer is shown in *Appendix A*, with the complete VHDL source code for the CY7C371 CPLD listed in *Appendix B*. This multiplexer allows eight RS-232 interfaces to be combined into a single high-speed serial stream. Because of the sampling-based architecture, each RS-232 interface can operate as synchronous or asynchronous, and each can also operate at totally independent data rates.

The complete design for one end of the multiplexer can be built from a single clock oscillator and five ICs: two MAX208 RS-232 level converters, one CY7C371 CPLD, a CY7B923 HOTLink transmitter, and a CY7B933 HOTLink receiver. With the components, the entire multiplexer operates from a single +5VDC supply and dissipates less than 2W.

Standard RJ-11-type connectors are used for the RS-232 connections. While not an official standard connector for RS-232 interfaces, it has found common industry usage for serial interfaces due to its low cost and high packing density.

The Cypress CY7C371 Flash programmable CPLD is a perfect fit for this application. The multiplexer design makes use of every available pin on the part. Supporting dual clocks and 32 macrocells, the designs for the sampling registers, output register, reframe control state machine, and metastable conversion functions, all fit with fully automatic pin placement using the Cypress *Warp2*® VHDL compiler, all without any special compiler directives.

The Cypress CY7B923/933 HOTLinks are only shown interfacing to the CY7C371 CPLD. Both parts are configured in ENCODED mode to allow use of their integrated 8B/10B encoder/decoders. The HOTLink receiver is also configured with RF hardwired HIGH to force the receiver's framer into multi-byte framing operation.

The design and implementation of various high-speed serial interfaces are not shown here. Detailed information on how to interface to various types of copper or optical media may be found in the Cypress *HOTLink User's Guide*.

The maximum serial rate supported by this design is limited by the MAX208 RS-232-to-TTL converters. These parts provide the level conversion from RS-232-to-TLL and TTL-to-RS-232 signal levels. Since the maximum data rate supported by these level converters is 120 kbits/second, the oscillator used to clock the sampling registers and the HOTLink parts can be run as slow as 16 MHz. Even at this slow clock rate, each RS-232 interface is sampled at over 100 times its maximum bit rate.

Because of the high sampling rate, this design could easily be expanded to 16, 32, or even 64 serial lines with only a slight modification to the control circuitry. This would involve replication of the level translators for all added ports as well as the sampling and output registers.

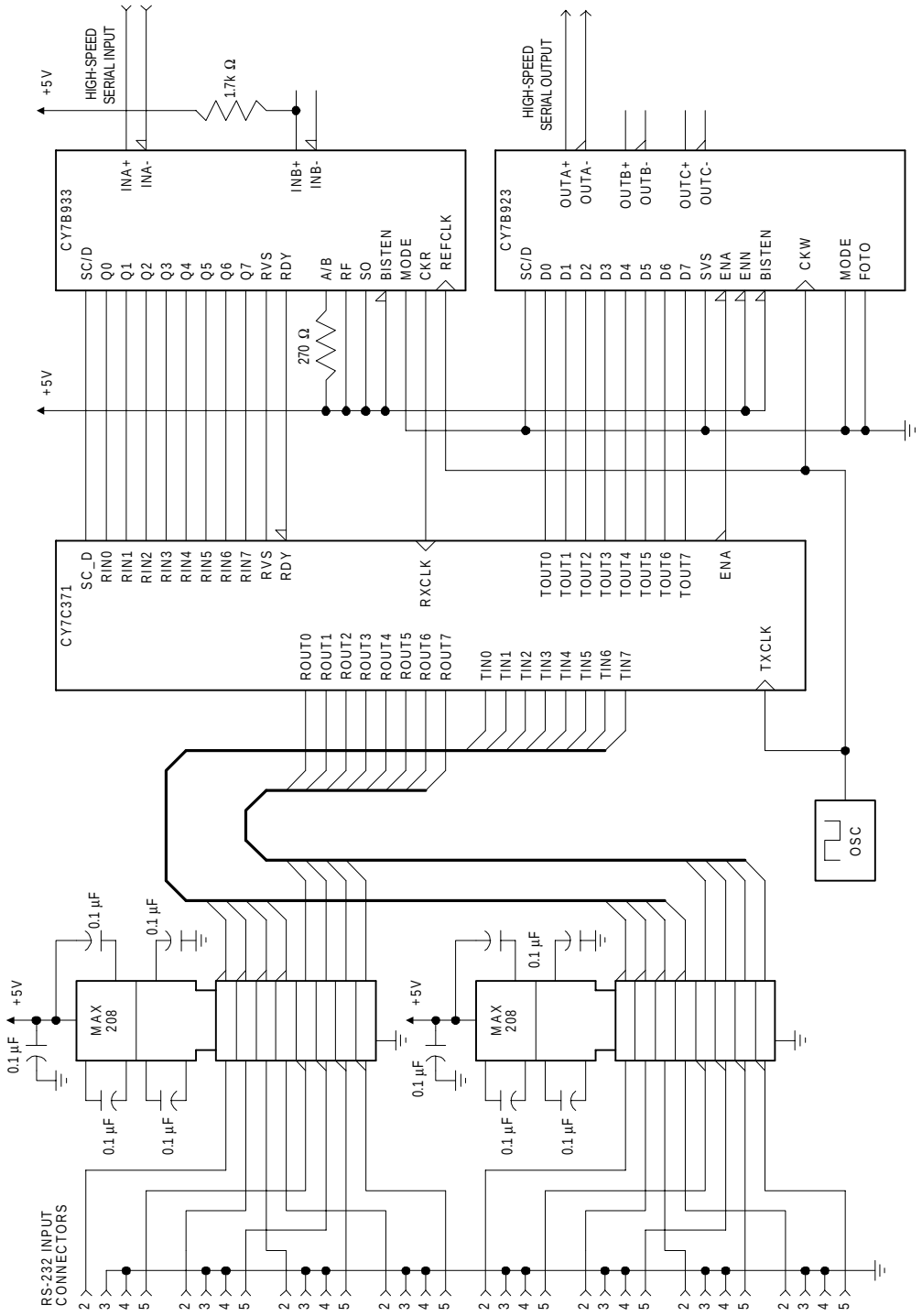
In addition to the level translators, a pair of small counters (3-bit for 64 serial lines) would be added to each end of the high-speed link. These counters would control the sequencing of each of the sampling registers to the HOTLink transmitter, and similar sequencing of the captured data to the output registers.

For this sequencing to work correctly, these counters must both be initialized to the same count. This initialization can be handled by the presence of a received SYNC character, such that the counter at the transmit end of each link is reset whenever $\overline{\text{ENA}}$ is driven HIGH (forcing transmission of a SYNC character), and the counter at the receive end of each link is reset when any SYNC character is detected.

Conclusion

Multiplexing multiple signals onto a high-speed serial link often allows connections between pieces of equipment to be made in a more economical fashion than routing the individual signals as a parallel bus. The RS-232 serial multiplexer design in this application note can be easily modified to support other serial or parallel interfaces (or a mixture of these). This can allow an interface to be extended from what would normally be a few meters, to multiple kilometers, with minimal amounts of hardware and cost.

Appendix A. Eight-Port RS-232 Serial Multiplexer Schematic



Appendix B. HOTLink Serial Multiplexer CPLD Source Code

```
-- SER_MUX.VHD

-- Serial interface multiplexer. This design uses a synchronous PLD to
-- both perform as a metastable prevention circuit for loading asynchronous
-- slow-speed serial data into a HOTLink transmitter, and for controlling
-- automatic framing functions when framing is lost at a receiver.

-- Mode of automatic recovery is
-- 1. if errors are detected (RVS indication) enter error recovery
-- 2. in error recovery, if you detected errors send a pair of
--    adjacent k28.5's for every error you detect
-- 3. if you are not in an error detect state (state1) and receive,
--    k28.5s, send a pair for every burst you detect, but with a byte
--    between each k28.5

ENTITY ser_mux_top IS PORT (
        txclk: IN BIT;                  -- HOTLink transmitter byte clock
        rxclk: IN BIT;                  -- HOTLink receiver recovered clock
        tin: IN BIT_VECTOR(0 TO 7);     -- asynchronous serial data bits in
        tout: OUT BIT_VECTOR(0 TO 7);   -- synchronous serial data bits out
        rin: IN BIT_VECTOR(0 TO 7);     -- synchronous serial data bits in
        rout: BUFFER BIT_VECTOR(0 TO 7); -- synchronous serial data bits out
        SC_D: IN BIT;                   -- HOTLink RX SC/D pin
        RDY: IN BIT;                    -- HOTLink RX /RDY signal
        RVS: IN BIT;                    -- HOTLink RX RVS signal
        ENA: OUT BIT                    -- HOTLink TX ENA signal
        );

ATTRIBUTE part_name OF ser_mux_top:ENTITY IS "C371";

END ser_mux_top;

USE work.cypress.all;

ARCHITECTURE serial_mux_top OF ser_mux_top IS

-- declare internal signals
SIGNAL txi: BIT_VECTOR(0 TO 7);-- first metastable pipeline register
SIGNAL RVS_rx1: BIT;            -- captured RVS signal
SIGNAL RVS_tx1: BIT;            -- RVS in txclk domain
SIGNAL RVS_tx2: BIT;            -- RVS after second metastable latch
SIGNAL RDY_rx1: BIT;            -- captured RDY signal
SIGNAL RDY_tx1: BIT;            -- RDY in txclk domain
SIGNAL RDY_tx2: BIT;            -- RDY after second metastable latch

-- declare state machine
TYPE sync_state IS (
        state0,                 -- no errors, waiting for RVS or RDY
                                --  output /ENA (LOW)
        state1,                 -- RVS or RDY found, output ENA (HIGH)
```

Appendix B. HOTLink Serial Multiplexer CPLD Source Code (continued)

```
        state2,                 -- RDY active, output /ENA (LOW)
        state3                  -- output ENA (HIGH)
        );

-- declare state machine encoding, state variable, and initial state
SIGNAL s_state : sync_state := state0;

BEGIN

-- Pipeline all eight data bits to the HOTLink TX
meta1: PROCESS BEGIN
        WAIT UNTIL txclk = '1';
                tout <= txi;
                txi <= tin;
        END PROCESS meta1;

-- Filter out command codes from receive data
rxfilt1: PROCESS BEGIN
        WAIT UNTIL rxclk = '1';
        IF (SC_D = '0') THEN
                rout <= rin;
        ELSE
                rout <= rout;
        END IF;
        END PROCESS rxfilt1;

-- latch and synchronize RVS signal
rxfilt2: PROCESS BEGIN
        WAIT UNTIL rxclk = '1';
        IF (RVS = '1') THEN
                RVS_rx1 <= '1';
        ELSIF (RVS_tx2 = '1') THEN
                RVS_rx1 <= '0';
        ELSE
                RVS_rx1 <= RVS_rx1;
        END IF;
        END PROCESS rxfilt2;

-- latch and synchronize RDY signal
rxfilt3: PROCESS BEGIN
        IF (RDY = '1') THEN
                RDY_rx1 <= '1';
        ELSIF (RDY_tx2 = '1') THEN
                RDY_rx1 <= '0';
        ELSE
                RDY_rx1 <= RDY_rx1;
        END IF;
        END PROCESS rxfilt3;
```

Appendix B. HOTLink Serial Multiplexer CPLD Source Code (continued)

```
-- add synchronization flops
rx_tx_filt: PROCESS BEGIN
        WAIT UNTIL txclk = '1';
        RDY_tx2 <= RDY_tx1;
        RDY_tx1 <= RDY_rx1;
        RVS_tx2 <= RVS_tx1;
        RVS_tx1 <= RVS_rx1;
        END PROCESS rx_tx_filt;

-- monitor receive path for violation errors
proc1:  PROCESS BEGIN
        WAIT UNTIL (txclk='1');
        CASE s_state IS
                WHEN state0 =>
                        IF ((RVS_tx2 = '1') OR (RDY_tx2 = '1')) THEN
                                s_state <= state1;
                        ELSE
                                s_state <= state0;
                        END IF;
                WHEN state1 =>  -- RVS or RDY detected
                        IF (RVS_tx2 = '1') THEN
                                s_state <= state3;
                        ELSE
                                s_state <= state2;
                        END IF;
                WHEN state2 =>  -- RDY detected
                                s_state <= state3;
                WHEN state3 =>
                        IF (RVS_tx2 = '1') THEN
                                s_state <= state3;      -- keep ENA HIGH
                        ELSIF (RDY_tx2 = '1') THEN
                                s_state <= state2;
                        ELSE
                                s_state <= state0;
                        END IF;
                WHEN others =>
                        s_state <= state0;
        END CASE;
END PROCESS proc1;

-- assign outputs
-- force k28.5 codes when errors have been detected
ENA <= '1' WHEN (s_state = state1 OR s_state = state3) ELSE '0';

END serial_mux_top;
```

HOTLink is a trademark and *Warp2* is a registered trademark of Cypress Semiconductor Corporation.