Port Error Recovery and Resynchronization

Application Note

80B803A_AN003_02

August 5, 2009

6024 Silver Creek Valley Road San Jose, California 95138
Telephone: (408) 284-8200 • FAX: (408) 284-3572
Printed in U.S.A.

GENERAL DISCLAIMER

Integrated Device Technology, Inc. ("IDT") reserves the right to make changes to its products or specifications at any time, without notice, in order to improve design or performance. IDT does not assume responsibility for use of any circuitry described herein other than the circuitry embodied in an IDT product. Disclosure of the information herein does not convey a license or any other right, by implication or otherwise, in any patent, trademark, or other intellectual property right of IDT. IDT products may contain errata which can affect product performance to a minor or immaterial degree. Current characterized errata will be made available upon request. Items identified herein as "reserved" or "undefined" are reserved for future definition. IDT does not assume responsibility for conflicts or incompatibilities arising from the future definition of such items. IDT products have not been designed, tested, or manufactured for use in, and thus are not warranted for, applications where the failure, malfunction, or any inaccuracy in the application carries a risk of death, serious bodily injury, or damage to tangible property. Code examples provided herein by IDT are for illustrative purposes only and should not be relied upon for developing applications. Any use of such code examples shall be at the user's sole risk.

The IDT logo is registered to Integrated Device Technology, Inc. IDT and CPS are trademarks of Integrated Device Technology, Inc.

.

“Accelerated Thinking” is a service mark of Integrated Device Technology, Inc.

Port Error Recovery and Resynchronization

This document provides the RapidIO system developer with the knowledge to understand when software assisted error recovery is required and the mechanisms used to recover from an error. It also details an example implementation for handling errors.

This application note discusses the following:

- "Overview" on page 3
- "Serial RapidIO PCS and LP-Serial Protocol" on page 4
- "Sample Link Recovery Implementation" on page 9

Revision History

80B803A_AN003_02, Final, August 2009

Changes to this document include:

- Grammatical errors were fixed and some sentences were clarified
- The PORT_OK bit definition was added to the sample code for clarity
- Formatting updated for IDT

80B803A_AN003_01, Final, January 2009

This is the first version of this document.

1. Overview

Error recovery must be part of reliable system design. The Serial RapidIO protocol provides automatic error recovery in most circumstances. However, in some cases, software is expected to initiate error recovery. These cases include:

- Device reset
- Hot replacement of a system component
- Loss of a reliable physical connection between devices

Many AMCs, and similar modules, use IDT switches because of their error recovery functions.

2. Serial RapidIO PCS and LP-Serial Protocol

The Physical Coding Sublayer (PCS) and LP-Serial Protocol are low level standards implemented in the physical layer of Serial RapidIO devices. The PCS describes link initialization and the maintenance of a link using idle sequences. The LP-Serial Protocol describes the use of control symbols to manage a link, and the reliable delivery of packets across the link.

The information in this document is based on the *RapidIO Specification (Revision 1.3): Part 6: Serial Physical Layer Specification*. For more details about the layers not directly related to error recovery, including link initialization state machine diagrams, character definitions, and packet flow control negotiation, refer to the *RapidIO Specification (Revision 1.3)*.

The following sections describe the operation of the two layers under software assisted error recovery conditions.

2.1 PCS and LP-Serial Protocol Under Normal (Error Free) Conditions

When two Serial RapidIO devices power-up, they undergo link initialization which is defined by the PCS layer. In this phase, each end of the link determines the width capabilities of its partner, and aligns the transmission of bits so that communication may proceed between the two. After initialization, the link is maintained by the exchange of idle sequence characters across the link. Idle sequences are transmitted whenever the LP-Serial Protocol layer is not sending a control symbol or packet. These characters ensure that each end of the link maintains 8b/10b boundary synchronization, clock synchronization, and similar low-level parameters.

The LP-Serial layer is responsible for the reliable transmission of packets, and for link maintenance at a higher level than PCS. It performs these tasks through the use of control symbols. These symbols perform a number of functions, the most important being packet delimitation and error recovery.

As packets are exchanged from one end of the link to the other, control symbols are sent indicating the start and end of the packet. After the packet has been received, a control symbol is sent from the receiver indicating packet acknowledgement. As part of this acknowledgement, the receiver transmits an ackID which indicates which packet is being acknowledged. It is the responsibility of each end of the link to keep track of which ackIDs to acknowledge, and which ackIDs to expect acknowledgement for.

The RapidIO protocol allows for acknowledgement of multiple outstanding packets, packet retries, buffer status reporting, and similar performance improving mechanisms. These concepts, and their implementation, are beyond the scope of this document but are described in detail in the *RapidIO Specification (Revision 1.3)*.

2.2 LP-Serial Protocol Under Error Conditions

Although the LP-Serial Protocol handles many errors automatically, certain error conditions result in the port entering an error-stopped state. These errors include: invalid control symbols, invalid idle sequences, CRC errors, and acknowledgement timeouts.

The most common cause of these errors is a loss of connectivity which may be caused by the power down of one link partner, the reset of a link partner, or the disconnect of a cable. A receiver enters the input error-stopped state when an error is detected, while a transmitter enters the output error-stopped state. While in a stopped state, a port neither sends nor receives packets, but does allow the exchange idle sequences and control symbols if a physical link is present.

The error-stopped states can be accompanied by a loss of ackID synchronization between the two link partners. This is often the case when one device is reset or replaced, causing internal registers to revert to their power-on values. Recovering from a link failure is a two step process: recovering from the error-stopped state, and realigning the ackIDs. The error-stopped states may be recovered from by using a defined set of control symbols (detailed in the *RapidIO Specification (Revision 1.3)*). After the stopped states have been cleared, the ackID recovery procedure can be followed to bring the link to a fully operational state.

2.2.1 Recovering from Error-stopped States

Recovery from error-stopped states is performed by following the process in the *RapidIO Specification (Revision 1.3)*. A summary of the procedure to recover from stopped states is shown in Figure 1. The procedure involves the exchange of several control symbols, most of which is handled automatically by the hardware.

Figure 1: Error-Stopped State Recovery Process

	Endpoint A	Endpoint B
Step 1	PORT_OK	PORT_OK
Step 2	Reset, PORT_UNINIT	
Step 3		PORT_UNINIT, Enters IES
Step 4	PORT_OK	PORT_OK, IES
Step 5		Transmits Control Symbol
Step 6	PORT_OK, Enters OES	
Step 7	Transmits Control Symbol	
Step 8		PORT_OK, Exits IES
Step 9		Transmits Control Symbol
Step 10	PORT_OK, Exits OES	
Step 11	PORT_OK	PORT_OK

Figure 1 Description

1. Both link partners are in normal state, and able to exchange packets
2. Error occurs as endpoint A is reset
3. Endpoint B enters IES state in response to link going down unexpectedly
4. Link re-established and PORT_OK is seen on either side
5. Endpoint B generates a packet-not-accepted/link-request input-status control symbol to endpoint A.
   - The generation of this control symbol is the only action which needs to be initialized by software
     - An example of the register-write performed is in the RIO_Err_Clear.txt script, which is included with the IDT JTAG software which is available at www.IDT.com. Once the endpoints are out of their error-stopped states (in step 10 of this process), other actions may need to be performed to get the system into a normal state again. For example, error registers may need to be cleared, and ackIDs may need to be realigned (see "Realigning ackIDs").
6. Endpoint A enters OES because of the packet-not-accepted control symbol
7. Endpoint A transmits an link-response/link-request input-status (restart-from-error) control symbol
8. Endpoint B receives the control symbol and exits IES
9. Endpoint B transmits link-response control
   - Endpoint B can now send and receive packets
10. Endpoint A receives the link-response and exits OES state
11. Both endpoints can now send and receive packets

2.2.2 Realigning ackIDs

Each end of the link responds to a packet by generating control symbols acknowledging the packet's receipt through ackIDs (discussed in "PCS and LP-Serial Protocol Under Normal (Error Free) Conditions").

The following ackID values are tracked by each partner:

- Inbound: The ackID of the next, incoming packet. When a packet is received, the receiver replies with this value to acknowledge the packet.
- Outstanding: The ackID of the first packet which has been transmitted but not yet acknowledged. The transmitter will expect the next acknowledgment to contain this value.
- Outbound: the ackID associated with the next packet to be transmitted.

Each link partner stores the ackID values in the Port n Local ackID CSR. If all packets between two points have been received and acknowledged, the inbound field of each partner will match both the outstanding and outbound fields of the other partner.

2.2.2.1 Error Conditions

In some error conditions, the ackID values stored by one endpoint may not match the values stored by the other. This is most common when one partner is reset or replaced, initializing its internal register to contain 0 for each field (as shown in Figure 1 where endpoint A was reset). While the procedure in Figure 1 does exit the error-stopped states, the first packet transmitted will likely be refused by the receiver because it was deemed to have an incorrect ackID, causing IES to be reentered.

The procedure to align ackIDs can be performed by software that is able to access one end of the misaligned link.

Figure 2: Re-alignment of ackIDs

	Endpoint A			Endpoint B		
	Inbound	Outbound	Outstanding	Inbound	Outbound	Outstanding
Step 1	B	A	A	A	B	B
Step 2	C	D	E	A	B	B
Step 3	C	D	E	A	B	B
Step 4	C	D	E	A	C	C
Step 5	C+1	A	A	A	C+1	C+1

Figure 2 Description

1. The ackIDs are aligned
2. An error occurs on endpoint A, causing each field to become an unexpected value
   - If endpoint A was reset, each value (C, D, and E) is 0
3. Endpoint B sends a link-request input-status control symbol

This step was also performed in the "Recovering from Error-stopped States" process.

   - A second link-request input-status symbol does not need to be generated if performing ackID alignment immediately after error-stopped state recovery because the ackIDs would not have changed
   - Endpoint A responds with a link-response control symbol which indicates its next expected incoming ackID
4. Endpoint B reads the inbound ackID from bits 22-26 of the Port n Link Maintenance Response CSR. It then programs the outstanding and outbound fields of its own ackID register to reflect this value.

— Endpoint A is now able to receive packets

5. Endpoint B uses a maintenance write to program its partners Port n Local ackID CSR. The value written includes C+1 as the inbound ackID, and A as the outstanding and outbound ackIDs. The reason for this is that as the maintenance write is sent with ackID C, endpoint B increments its outbound field to C+1 and expects ackID C to be acknowledged from endpoint A. Endpoint A acknowledges C and increments its inbound ackID. The register write is then performed on endpoint A and the C+1 value is retained.

   — Packets can now be exchanged normally

3. Sample Link Recovery Implementation

The following code sample shows a routine which can be run on a host processor in a RapidIO system in order to recover from an error condition. The theoretical system used in this sample includes a Tsi578 which is accessed from the host processor using maintenance transactions. The code can be used to recover from an error-stopped state on any port (other than the port the host is connected to) and the realigns ackIDs on the port.

The port number of the device on the opposite end of the link is also required for this code. It is 0 if connected to an endpoint with a single RapidIO port, but it can also be another value if the Tsi578 is connected to another switch.

The routine has the following functionality:

1. Verifies PORT_OK is seen, and exits if PORT_OK is not seen
2. Locks out the port for a brief interval by asserting (and clearing) PORT_LOCKOUT bit
   - This causes any queued packets to be flushed, preventing corruption while aligning ackIDs
3. Uses the IDT-specific RIO Port x Control Symbol Transmit register to recover from error-stopped state (see "Recovering from Error-stopped States")
4. Re-aligns the ackIDs (see "Realigning ackIDs")
5. Clears error flags in per-port registers
   - This is not discussed in this document, and is not strictly necessary for the resumption of packet exchange

It is assumed that the following functions are implemented by the user:

`unsigned int Tsi578MaintRead(unsigned int offset)`

- Performs a maintenance read on the switch, returning the value of the specified register

`void Tsi578MaintWrite(unsigned int offset, unsigned int value)`

- Performs a maintenance write on the switch, writing the specified register with the specified value

`void PartnerMaintWrite(unsigned int offset, unsigned int value)`

- Performs a maintenance write on the link partner

`void ClearErrorRegisters(int port, int remote_port)`

- Clears any per-port error registers, both on the Tsi578 and the link partner, which may have had errors flagged
- The error bits can also be reported, or logged, for future reference (optional but recommended)

Figure 3: Code to Recover from Input Error Stop State

```
#define CS_PNA_LREQ 0x40fc8000 // packet-not-accepted/link-request
#define PORT_OK 0x00000002 // PORT_OK bit
#define ClearErrorRegisters(_port_, _remote_port_) // empty definition

void Tsi578SynchronizePort(int port, int remote_port)
{
   unsigned int control;
   unsigned int err_and_status;
   unsigned int our_ack_id_stat;
   unsigned int partner_ack_id_stat;
   unsigned int inbound;
   unsigned int outbound;

   /* Only run if PORT_OK */
   err_and_status = Tsi578MaintRead(Tsi578_SPX_ERR_STATUS(port));
   if (!(PORT_OK & err_and_status))
      return;

   /* Remove any queued packets by locking/unlocking port */
   control = Tsi578MaintRead(Tsi578_SPX_CTL(port));
   Tsi578MaintWrite(Tsi578_SPX_CTL(port), (control | 0x2)); //set bit
   delay(1000); // delay 1us
   Tsi578MaintWrite(Tsi578_SPX_CTL(port), (control & ~0x2)); //clear bit

   /* Send control symbol to recover from IES/OES */
   Tsi578MaintWrite(Tsi578_SPX_CS_TX(port), CS_PNA_LREQ);

   /* find out what ackID partner is expecting (0 in a rebooted device) */
   partner_ack_id_stat = Tsi578MaintRead(Tsi578_SPX_LM_RESP(port));
   outbound = (partner_ack_id_stat & 0x000003e0) >> 5;

   /* Align our outbound/outstanding ackIDs with partner's inbound */
   our_ack_id_stat = Tsi578MaintRead(Tsi578_SPX_ACKID_STAT(port));
   inbound = (our_ack_id_stat & 0x1f000000) >> 24;
   our_ack_id_stat = (inbound << 24) | (outbound << 8) | outbound;
   Tsi578MaintWrite(Tsi578_SPX_ACKID_STAT(port), our_ack_id_stat);

   /* Align partner's outbound/outstanding ackID with our inbound */
   outbound++;
   partner_ack_id_stat = (outbound << 24) | (inbound << 8) | inbound;
   PartnerMaintWrite(SPx_ACKID_STAT(remote_port), partner_ack_id_stat);

   /* Clear error bits and interrupts on each side of the link */
   ClearErrorRegisters(port, remote_port);
}
```