

Introduction

This document describes the operation of the interrupt controller for the Excalibur™ devices, particularly the different interrupt modes available when using the interrupt controller.

The interrupt controller interprets the six interrupt signals from the PLD to the processor subsystem and works in any of the following three interrupt modes:



EXCALIBUR™

- Six individual interrupts
- Six-bit priority scheme
- Five-bit priority scheme plus one individual interrupt



This document assumes that you are familiar with the architecture of Excalibur devices, which is detailed in the *Excalibur Devices Hardware Reference Manual*.

Operating Modes

The interrupt controller provides a simple and flexible software interface to the interrupt system. It generates two interrupt signals to the processor, INT_FIQ_n and INT_IRQ_n, formed from the following 17 interrupt sources:

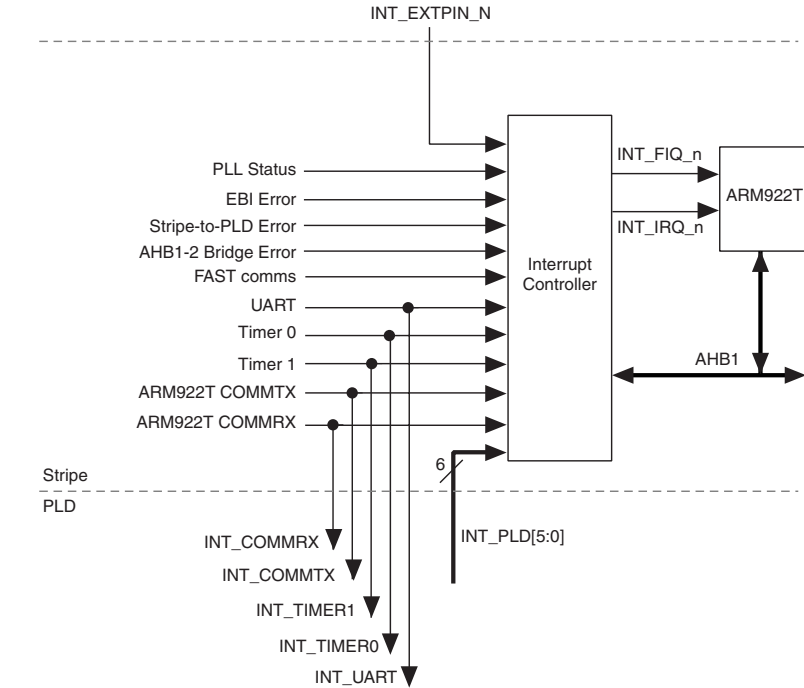
- 10 interrupts from modules inside the stripe
- 1 external pin
- 6 from the PLD-stripe interface



In addition, five of the stripe module interrupts are available as inputs to the PLD. Therefore, you can implement another interrupt controller in the PLD using the five interrupt signals and the six INT_PLD[5:0] signals.

Figure 1 on page 2 shows the signals processed by the interrupt controller.

Figure 1. Signals Processed by the Interrupt Controller



Three interrupt operating modes control how the six INT_PLD[5:0] signals are interpreted. The interrupt mode is determined by setting the INT_MODE register and is one of the following modes:

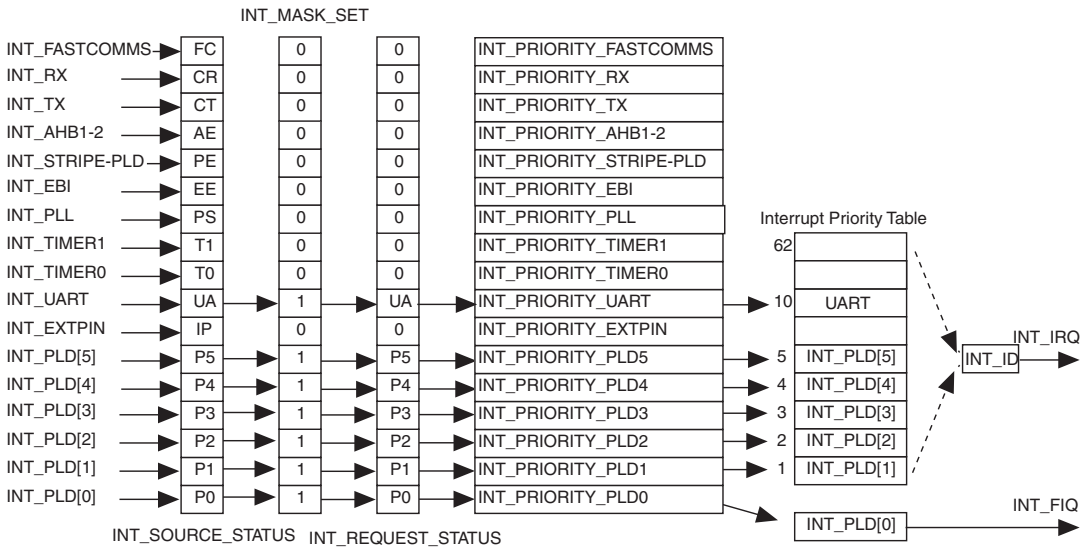
- Six individual interrupts (default)
- Single interrupt request using a six-bit priority value
- Single interrupt request using a five-bit interrupt priority value and one individual interrupt

Six Individual Interrupts Mode

The six individual interrupts mode is the default mode at system reset. It is typically used when six or fewer interrupts from the PLD are required.

Each PLD interrupt signal has its own mask bit in `INT_MASK_SET`; setting its corresponding mask bit in this register enables it to interrupt the embedded processor. In addition, each PLD interrupt signal has its own mask bit in `INT_MASK_CLEAR`; setting its corresponding mask bit in this register disables it from interrupting the processor. In the example design, the UART and `INT_PLD[5 : 0]` interrupt signals are enabled by setting the relevant mask bits as shown in [Figure 2 on page 3](#).

Figure 2. Six Individual Interrupts Mode



The `INT_SOURCE_STATUS` and `INT_REQUEST_STATUS` registers provide the status of each interrupt source before and after masking. In the example design, only the UART and the `INT_PLD[5 : 0]` interrupts can be monitored by the `INT_REQUEST_STATUS` register.

Each of the `INT_PLD[5..0]` interrupts has its own priority register (`INT_PRIORITY_PLD0` to `INT_PRIORITY_PLD5`); each can have a unique priority value between 1 and 62 for the `PRI` bit field, or have its `FQ` bit set to generate an `FIQ` interrupt. In the example design, interrupts `INT_PLD[1]` to `INT_PLD[5]` are assigned interrupt priority values of 1 to 5, respectively; while the stripe UART is assigned a priority value of 10. If more than one of the `INT_PLD[5:1]` interrupts occur at the same time, the `INT_ID` register contains the highest priority interrupt and an `IRQ` interrupt is generated to the processor. For example, if `INT_PLD[5]` (priority 5) and `INT_PLD[1]` (priority 1) signals are all pending, `INT_PLD[5]` is serviced first. However, the `INT_PLD[0]` signal is set to generate an `FIQ` interrupt, which always triggers independently of the `IRQ` interrupt.

Six-Bit Priority Value Mode

When more than six interrupt sources are required from the PLD, six-bit priority value mode can be used. In this mode, up to 63 interrupts can be encoded by user logic in the PLD. The `INT_PLD[5:0]` signals are interpreted as a six-bit encoded interrupt priority with either of the following values:

- 0—no interrupt from the PLD
- Non-zero value—a requesting interrupt with priority ranging from 1 to 63.

You must implement your own interrupt controller in the PLD, because the following bits and registers have no effect in this mode:

- Bits P5 to P0 of `INT_MASK_SET/INT_MASK_CLEAR`
- Bits P5 to P0 of `INT_SOURCE_STATUS`
- Bits P5 to P0 of `INT_REQUEST_STATUS`
- `INT_PRIORITY_PLD5` to `INT_PRIORITY_PLD0`

In the example design, a simple interrupt controller is implemented by using a priority encoder to determine the priority of the PLD interrupts and a mask register to enable the PLD interrupts. Nine interrupt sources (`INT0` to `INT8`) in the PLD first go through a mask register (implemented as a PLD slave register) to allow individual sources to be selectively passed through by setting its mask bits. Then, the priority encoder takes the nine interrupt sources and outputs a four-bit encoded value for the highest-priority interrupt that is pending; [Table 1 on page 5](#) shows the priority scheme. For example, if interrupts `INT0` to `INT8` occur at the same time, `INT_PLD[5:0]` will contain the value 9 (`INT_PLD[5:4]` is set to 0 in this design).

Table 1. Truth Table of 10-to-4 Priority Encoder

Input									Output			
INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	INT_PLD[3]	INT_PLD[2]	INT_PLD[1]	INT_PLD[0]
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	X	0	0	1	0
0	0	0	0	0	0	1	X	X	0	0	1	1
0	0	0	0	0	1	X	X	X	0	1	0	0
0	0	0	0	1	X	X	X	X	0	1	0	1
0	0	0	1	X	X	X	X	X	0	1	1	0
0	0	1	X	X	X	X	X	X	0	1	1	1
0	1	X	X	X	X	X	X	X	1	0	0	0
1	X	X	X	X	X	X	X	X	1	0	0	1

Legend:

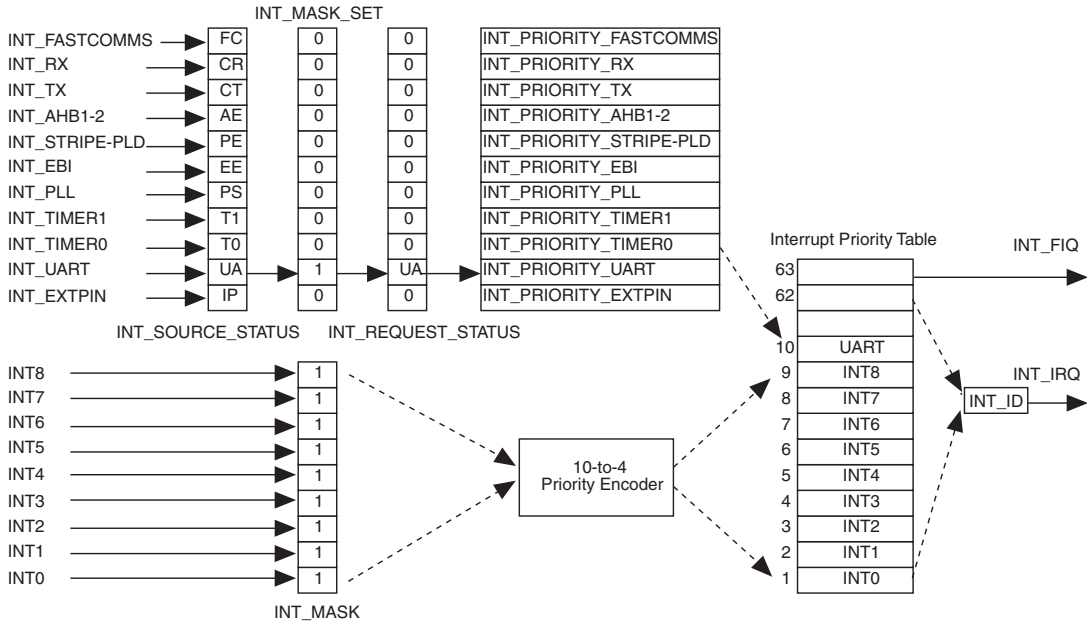
0—False

1—True

X—Don't care

The encoded `INT_PLD[5:0]` value is then compared with other requesting interrupt priorities from modules in the processor subsystem, to produce an IRQ or FIQ request, as shown in [Figure 3 on page 6](#). After the comparison, the interrupt identity register, `INT_ID`, contains the priority of the highest interrupt priority that is currently pending.

Figure 3. Six-Bit Priority Interrupts



Five-Bit Priority Value Plus Individual Interrupt Mode

In the five-bit priority value plus individual interrupt mode, INT_PLD[5] to INT_PLD[1] are treated as the most-significant bits of a six-bit encoded interrupt priority and INT_PLD[0] is treated as an individual interrupt. The least-significant bit is always 0, with either of the following values:

- 0—no interrupt from the PLD
- Non-zero even value—a requesting interrupt with priority range between 2 and 62.

Users must implement their own interrupt controller in the PLD, because the following bits and registers have no effect in this mode:

- Bits P5 to P1 of INT_MASK_SET/INT_MASK_CLEAR
- Bits P5 to P1 of INT_SOURCE_STATUS
- Bits P5 to P1 of INT_REQUEST_STATUS
- INT_PRIORITY_PLD5 to INT_PRIORITY_PLD1

In the example design, a simple interrupt controller is implemented by using a priority encoder to determine the priority of the PLD interrupts and a mask register to enable the PLD interrupts. Nine interrupt sources (INT1 to INT9) in the PLD initially go through a mask register (implemented as a PLD slave register) to allow individual sources to be selectively passed through by setting its mask bits. Then, the priority encoder takes the nine interrupt sources and outputs a four-bit encoded value for the highest priority interrupt that is pending (Table 2 shows the priority scheme). For example, if INT1 to INT9 all occur at the same time, the six-bit encoded value will be 18 (INT_PLD[5] is fixed to 0 in this design and the least-significant bit is always 0). The INT_PLD[0] signal is set to generate an FIQ interrupt to the processor by setting the FQ bit in the INT_PLD_PRIORITY_PLD0 register.

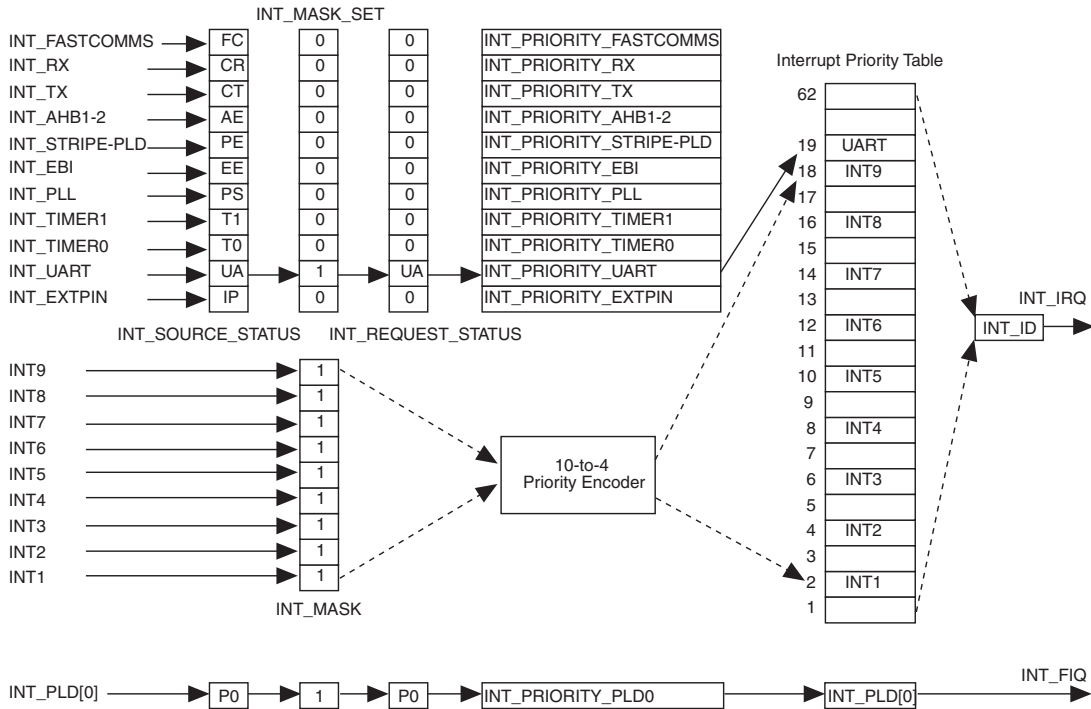
Table 2. Truth Table of 10-to-4 Priority Encoder

Input									Output			
INT9	INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT_PLD[4]	INT_PLD[3]	INT_PLD[2]	INT_PLD[1]
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	X	0	0	1	0
0	0	0	0	0	0	1	X	X	0	0	1	1
0	0	0	0	0	1	X	X	X	0	1	0	0
0	0	0	0	1	X	X	X	X	0	1	0	1
0	0	0	1	X	X	X	X	X	0	1	1	0
0	0	1	X	X	X	X	X	X	0	1	1	1
0	1	X	X	X	X	X	X	X	1	0	0	0
1	X	X	X	X	X	X	X	X	1	0	0	1

Legend:
 0—False
 1—True
 X—Don't care

The six-bit encoded value is then compared with other requesting interrupt priorities from modules in the processor subsystem, to produce an IRQ or FIQ request as shown in Figure 4 on page 8. After the comparison, the interrupt identity register, INT_ID, contains the priority of the highest-priority interrupt that is currently pending.

Figure 4. Five-Bit Priority Value Plus Individual Interrupt



Revision History

Table 3 shows the document revision history.

Date	Description
February 2002	First publication.
March 2002	Replaced <code>switch(INT_ID)</code> with <code>switch(irqID)</code> in example code.
September 2002	Example design modified and interrupt vector table section removed.
December 2002	Updates for the Quartus II software version 2.2.

The example design included with this application note contains the Quartus® II project files and all source code.



This appendix assumes that you are already familiar with the Excalibur toolflow: synthesizing the hardware design; compiling the software application; and programming the PLD image into the EPXA10 development board. If not, please refer to the *EPXA10 Development Board Getting Started User Guide* for details of the procedure.

Design Overview

The hardware portion of the design consists of binary counters that activate each PLD interrupt source at different intervals. Two PLD slave registers are used to clear and mask any active PLD interrupt sources. The software application processes the PLD interrupt sources, deactivates the interrupt source that is captured, and prints a message to a terminal window signifying which signal was serviced.

The example design consist of three projects (**individual**, **6_priority**, **5_priority**)—one project for each interrupt mode. To run the example design for a particular interrupt mode, open the corresponding project.

Building the Design

To build the example design that uses six individual interrupts mode (steps for building the other interrupt modes are similar), perform the following steps:

1. Start the **Quartus II** software.
2. Open **interrupt.quartus** in the **interrupt\individual\hardware** directory.
3. Click on **Start Compilation** (Processing menu) to build the hardware design.
4. Open **interrupt.quartus** in either the **interrupt\individual\ads** or **interrupt\individual\gnu** directory, depending on which compiler you are using.
5. Choose **Start Software Build** (Processing menu) to build the software design.

Alternatively, you can run the makefile in a Command Prompt window by typing `make` in the appropriate `interrupt\individual\ads` or `interrupt\individual\gnu` directory.

Running the Design

To run the example design, perform the following steps:

1. Configure the following jumpers on the EPXA10 development board, as specified in [Table 4](#).

Table 4. Jumper Configuration

Jumper	Position	Description
MSEL0	1-2	Boot from 16-bit flash
MSEL1	1-2	Boot from 16-bit flash
DEBUG_EN	2-3	Enable debug mode
BOOT_FLASH	2-3	Boot from flash
JP40	1-2	Select the external 32-MHz oscillator on the EPXA10 board for the PLD CLK0 source
JP51	2-3	Enable reference clock for EPXA10 stripe
JP54	2-3	Enable PLD CLK0

2. Start up a HyperTerminal window configured for 38400 baud rate, 8 data bits, no parity, 1 stop bit, and no flow control.
3. Configure the PLD with the example design by executing the `prog_hw.bat` DOS batch file in the `interrupt\individual\ads` or `interrupt\individual\gnu` directory. The software application program then displays the corresponding PLD interrupt being generated.



101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
<http://www.altera.com>
Applications Hotline:
(800) 800-EPLD
Literature Services:
lit_req@altera.com

Copyright © 2002 Altera Corporation. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services. All rights reserved.

